Dynamic Deployment of Mobile and Stationary Sensor Nodes for the Rapid Exploration of Emergency Areas

Ettore Ferranti, Niki Trigoni and Mark Levene

Birkbeck College, University of London

Grant FA8655-06-1-3003: Comprehensive Final Report

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
28-09-2007	Final Report	12 April 2006 - 12-Apr-07

4. TITLE AND SUBTITLE

Dynamic Deployment of Mobile and Stationary Sensor Nodes for the Rapid Exploration of Emergency Areas

5a. CONTRACT NUMBER
FA8655-06-1-3003

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Dr. Niki Trigoni

5d. PROJECT NUMBER

5d. TASK NUMBER

5e. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Oxford
Wolfson Building Parks Road
Oxford OX1 3QD
United Kingdom

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
PSC 821 BOX 14
FPO AE 09421-0014

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Grant 06-3003

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The study of distributed event processing is of major importance in security and surveillance applications. The project is envisioned to assist mission-critical operations by providing command and control systems, which collect sensor information and react promptly to the detection of suspicious high-level events. Events are monitored, composed and propagated within a wireless sensor network that consists of nodes with varying sensing capabilities, including cameras, microphones, magnetometers, temperature and infrared sensors.

Detecting complex events in a sensor network includes i) selecting a suitable event execution plan, i.e. event operators (e.g. event automata), and their order of execution, and ii) carefully placing event operators to physical nodes in the network.

This effort developed three novel search algorithms suitable for a distributed control environment, Multiple Depth First Search (MDFS), Brick and Mortar, and HybridExploration. They are effective in environments with the following assumptions: No a priori knowledge, lack of GPS positioning, and no centralized control of agents. The HybridExploration algorithm does make use of RFID tags and uses tag-to-tag communications, and assumes multi-hop communications as in a deployed sensor network.

15. SUBJECT TERMS
EOARD, Command and Control, Distributed sensors, Information Integrity, Information Assurance

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18, NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			PAUL LOSIEWICZ, Ph. D.
UNCLAS	UNCLAS	UNCLAS	UL	60	**19b. TELEPHONE NUMBER** *(Include area code)* +44 20 7514 4474

Effort sponsored by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA8655-06-1-3003. The U.S. Government is authorized to reproduce and distribute reprints for Government purpose notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.

Chapter 1

Introduction

When an emergency occurs within a building, it is crucial for the first responders to acquire as much information as possible on the ongoing situation, in order to identify and contain hazards and coordinate the rescue of victims. Initially, the area is off-limits and hazardous for anyone not wearing respiratory equipment, garments or barrier materials to protect themselves from exposure to biological, chemical, and radioactive hazards. This kind of suit can be very heavy and bulky, consequently limiting the first responders' movements, and reducing their sensing capacity (touch, vision, and hearing). A group of autonomous mobile nodes, referred to as *agents*, should therefore be deployed in the area to acquire all the information that could assist the tasks of the first responders.

Exploring all the area in the minimum amount of time and reporting back to the human personnel outside the building is an essential part of rescue operations. However, such operations may be obstructed by a number of limitations, e.g. the possible lack of a terrain map (the environment could in any case be substantially changed as the result of a disaster), the failure of previously established networks, and the short-range and often unreliable wireless indoor communication. In addition, it might be difficult to use GPS positioning inside a building, so an agent cannot rely on knowledge of its exact location within the terrain, even if it were able to memorise its previous steps.

In this work, we take into account these limitations, and assume that agents can rely only on local information that is sensed in their vicinity (which other agents have left behind them as a trace), before making the next exploration step. We propose a novel approach to area exploration in which a swarm of *agents* enter the emergency area and dynamically deploy a network of stationary sensor nodes, referred to as *tags*, in order to label the environment. Agents do not communicate directly with each other; instead, they coordinate indirectly by leaving traces of information on the tags that they deploy on the space. Agents are able to read and update the state of local tags, and by doing so, they leave valuable information for other agents in order to help them make intelligent

navigation decisions. In addition, the set of deployed tags form a multi-hop wireless network. Tags disseminate their local information to other tags downstream hop-by-hop in order to further assist the agents in their exploration task.

In this work we consider using two modes of communication: between agents and tags (agent-to-tag) and multi-hop communication within the stationary network of tags (tag-to-tag). Our goal is to design fully distributed algorithms that do not not require centralized control of the agents to determine their next move. The proposed algorithms only exploit short-range communication between agents and tags, and often among tags, and do not use unreliable long-range communication among agents, or agents and the human responders.

In this work, we present three novel algorithms, namely Multiple Depth First Search (MDFS) [1] (an adapted version of the DFS approach used also in [2, 3, 4]), Brick&Mortar [1], and HybridExploration algorithms. We also compare them against the existing literature showing the performance of the Ants algorithm by [5]. These algorithms make similar assumptions to those discussed above, i.e. no knowledge of the area map, lack of GPS positioning, and no centralized control of agents. Agents dynamically deploy tags on the floor and, by reading and updating the state of the local tags, they coordinate their exploration task. However, unlike the others, the HybridExploration algorithm does exploit tag-to-tag communication, i.e. it uses the multi-hop communication capabilities of the deployed sensor network.

During the development, we found out several weaknesses regarding the algorithms. For example, agents running the Ants algorithm cannot determine when the exploration task is completed. Moreover, whilst the first few agents rapidly discover new terrains, most of the remaining ones may dwell on already explored network areas, leading to inefficient use of agent resources. For instance, agents running MDFS know when the exploration task terminates, but their poor coordination leads to long exploration times. Finally, agents running Brick&Mortar use a complex loop resolution mechanism that significantly delays the task of area exploration, especially in topologies with many obstacles (e.g., desks in the middle of an open space). On the contrary, the last implemented HybridExploration algorithm overcomes all the limitations of previous approaches, and offers significant performance gains in terms of exploration time for a variety of terrain topologies. Our experimental results allow us to understand the impact of several parameters on the performance of the whole set of algorithms, including the number of agents, the terrain size, the numbers of rooms, and the number of obstacles (e.g., desks or hazards in the middle of rooms).

1.1 Model

We consider the task of exploring a hazardous terrain using a group of autonomous *agents*. The overall area is subdivided into square cells, some of them representing walls. In our model, walls are used to identify both real brick walls that constitute the building itself, and obstacles (e.g., furniture) that agents cannot cross during their exploration phase.

A cell can be in one of the following states:

- ***Wall***: The cell cannot be traversed by an agent because it is blocked by an obstacle, and any free space in it is not big enough to let an agent go past it.
- ***Unexplored***: No agent has been in the cell yet, and therefore no tag has been deployed there yet.
- ***Explored***: The cell has been traversed at least once, but the agents might need to go through it again in order to reach other *unexplored* cells. A tag is already deployed there by the agent who first visited the cell.
- ***Visited***: The agents have already explored the cell, and they do not need to go through it again to reach other cells. Conceptually, this state is equivalent to a *wall* cell, in that no agent is allowed to traverse it. A tag is already deployed in the cell.

In the remainder of this section, we provide a high level overview on the agent movement and tag deployment, and discuss the agent-to-tag and tag-to-tag modes of communication.

Agent movement and deployment of tags: Agents are initially deployed in one of the boundary cells and, in each step, they are able to move from the current cell to one of the four adjacent cells in the North, East, South or West directions. As they move to an unexplored cell, they deploy a miniature device (e.g., mote or RFID), referred to as *tag*, capable of storing small amounts of information about the state of the local cell. They also update the relative location of this tag, with respect to previously installed tags in adjacent cells. In indoor environments where GPS cannot be used, agents do not rely on knowledge of their exact location; once they find themselves within a cell, they can move towards one of the four directions until they reach the next cell.

Agent-to-tag communication: Moreover, in emergency situations, long-range wireless communication may be intermittent and unreliable, so we assume that agents are able to communicate only by reading and updating the tags installed in the local and adjacent cells. We thus only consider distributed exploration algorithms, in which agents make independent decisions about how to navigate through the terrain based on local state.

Tag-to-tag communication: In one of the proposed algorithm, HybridExploration, we introduce *virtual agents*, i.e. active messages that cause the execution of a small piece of

code on the tag that receives them. Virtual agents alter the state of the local tag and are further disseminated to neighboring tags within communication range. The underlying assumption of our model is that a tag in a cell is able to communicate wirelessly with the tags in the adjacent cells.

1.2 Objectives

We are now going to introduce two objectives, which we will use to assess the performance of the algorithms we are going to examine in Chapter 3. For each of the proposed algorithms, we will have an analysis in which the approach will be tested against its capability to achieve the two objectives. We are also going to use these two goals in Chapter 4, where the algorithms will be compared to each other according to their capabilities of achieving the objectives in several different scenarios.

1. **Exploration objective**: all cells in the area are traversed by a physical agent at least once. This means that no cell is left in the *unexplored* state. When this objective is achieved, cells can be in any of the *explored*, *visited* or *wall* states.

2. **Termination objective**: all cells in the area are either *walls* or *visited*. No cell is left in the *unexplored* or *explored* state. The marking and navigation steps of the Physical Agent Protocol allow agents to know when the termination objective is achieved based on local information about the state of adjacent cells. More specifically, it is easy to prove that when the four cells adjacent to the currently occupied cell in the North, East, South and West directions are either *wall* or *visited* cells, then the entire area consists entirely of *wall* and *visited* cells, and thus the termination objective has been achieved. Note that there is no corresponding *local rule* to notify agents when the exploration objective is achieved.

By definition, the exploration objective is always achieved earlier (or at the same time as) the termination objective. Both objectives should be achieved in the minimum amount of time, because in an emergency scenario as the one we are considering, speed is essential. The faster the exploration objective is achieved, the faster victims and hazards are identified. The faster the termination objective is achieved, the earlier human responders can enter the area with the certainty that there are no hidden hazards. The efficiency of an algorithm can be measured by how fast it is able to achieve both the exploration and termination objectives.

The goal of the present work is therefore to devise algorithms that achieve both the objectives expressed before, so that they can be used by robots operating in emergency scenarios.

The remainder of this dissertation is organized as follows:

Chapter 2 - Related Work
This chapter presents existing research on exploration algorithms subdividing them between *on-line* and *off-line* algorithms. We pay particular attention to describing the Ants algorithm [5] which makes the same assumptions and it is most related to this work. A final overview concerning the projects involved in the Robocup Rescue Competitions is provided at the end of the chapter.

Chapter 3 - Exploration Algorithms
This chapter describes several exploration algorithms designed to solve our research problem. Each of these algorithms represents an incremental step followed during the research process. At the beginning, we created the novel Multiple Depth First Search algorithm, then the Brick&Mortar one and finally, to overcome the weaknesses of the two competing approaches, we created the HybridExploration algorithm. In this chapter, for each algorithm we provide a detailed analysis and several explanatory examples.

Chapter 4 - Evaluation
This chapter presents the evaluation of the implemented exploration algorithms. Firstly, we describe the simulation environment created to test the algorithms, then we present the performances achieved by each exploration algorithm. A detailed analysis is provided to understand the impact of several parameters on the performance of the HybridExploration algorithm and the competing ones, including the number of agents, the terrain size, the numbers of rooms, and the number of obstacles. Finally, we demonstrate the feasibility of our approach in a real setting, using a robot that deploys Tmote Sky nodes on the floor of an open-plan office area.

Chapter 5 - Summary
This chapter summarizes the main contributions of our work.

Chapter 2

Related Work

In this chapter we will provide an overview of the existing exploration algorithms and the related research that has been carried out in the area. The existing approaches will be presented according to the assumptions they make, and one of the algorithms, Ants, will be presented with more details because it uses the same assumptions and the model that we discussed in Section 1.1, and it is therefore the best candidate for a comparison with our novel algorithms that will be presented in Chapter 3.

2.1 Exploration Algorithms Overview

Information gathering inside an area is essential to avoid risking the lives of the first responders: for example, if we knew the locations of the victims before entering a building, the responders could immediately get there avoiding hazardous areas such as rooms on fire or collapsed corridors or stairs. Exploring all the area in the minimum amount of time and reporting back to the human personnel outside the building is therefore an essential part of rescue operations. A group of mobile agents should therefore be deployed in the area to acquire all the information that could assist the tasks of the first responders. The existing algorithms used by the agents to perform the exploration task can be distinguished based on two criteria: (i) knowledge of map and (ii) communication modes.

1. **Knowledge of map**: Choset [6] provides a survey of coverage algorithms and distinguishes them into *off-line* and *on-line*. In the former, the agents are previously provided with a map of the area to explore, while in the latter, also called *sensor-based*, no assumption is made concerning the availability of an environmental map for the agents.

2. **Communication modes**: the robots communication can be classified in three main

different ways:

- *Agent-to-Server communication*: the robots coordination occurs through a central server;
- *Agent-to-Agent communication*: a direct wireless communication occurs between robots within the same range;
- *Agent-to-Environment communication*: our novel mode of communication, wherein agents communicate indirectly by interacting with an instrumented (smart) environment.

Our approach differs from related work in that we are investigating the subclass of *on-line* algorithms that rely on our novel mode of communication through the instrumented environment. To the best of our knowledge, little work [7, 5, 8, 1] has investigated the problem of *on-line* area exploration by letting agents coordinate indirectly by tagging the environment, subsequently reading and updating the state of the deployed tags. Moreover, the existing algorithms are not able to autonomously decide when the exploration is terminated. Recognising when the exploration terminates is of primary importance in an emergency scenario such as the one we are tackling: if the robots have to report back to the first responders the situation inside the building, they absolutely need to know when to stop exploring, and to do that they need to be sure whether all the area has been explored or not. Our approach is therefore aimed to propose algorithms which can autonomously explore the area and decide in the minimum amount of time when the task has been fulfilled.

A brief survey of existing *on-line* and *off-line* algorithms is provided in the two following subsections.

2.1.1 Off-line Algorithms

Since the *off-line* algorithms previously know the map of the environment, in theory they could build a set of optimal paths to cover the area using all the available agents in the minimum amount of time (optimal solution). In practice, this is not possible because this problem is NP-complete, therefore heuristics are needed to find a feasible solution in polynomial time. Such a heuristic is proposed in [9], where the authors prove that the original problem is NP-complete, and propose a polynomial algorithm, which runs in the worst case eight times slower than the optimal solution. Agmon et al. [10] propose a faster tree construction algorithm, while Hazon et al. [11, 4] focus on the robustness of the solution, so that even if only one robot remains in operation, it will be able to carry and complete the exploration task. All the *off-line* algorithms, being centralized in their computation of the exploration paths, use the Agent-to-Server communication mode.

2.1.2 On-line Algorithms

On-line algorithms should rely only on their sensors in order to navigate an unknown environment and be capable of taking *on-line* decisions about what to do next. Having many different possible environments, one algorithm could work better in an indoor environment with tiny rooms and a great number of corridors, while another could be faster in the case of big rooms interconnected by many doors, and so on. Most of these approaches divide the environment into cells, also called regions, that are explored one by one iteratively until the global area is covered.

Usually the *on-line* approaches assume that the agents are able to coordinate their movements using RF (radio frequency) communication. However, in the literature ([12], [13], [14], [15], [16], [17], and [18]) it is widely accepted that radio propagation is (i) non-isotropic (i.e., the received signal, at a given distance from the sender, is not the same in all directions), it has (ii) non-monotonic distance decay (i.e., lower distance does not mean better link quality), and (iii) the communication is based on asymmetrical links (i.e., if A hears B, it cannot be assumed that B hears A). For this reason, the agents coordination through RF (radio frequency) communication represents an assumption that is not always true.

In particular, we highlight the unreliability of RF communications in an indoor or underground environment in which the agents could be spread while carrying on the exploration task. The radio range of a wireless transceiver can in fact vary depending on the environment in which it is operated, and while if using a 2.4 GHz band we can easily have a 100 meters range outdoor, the same range is restricted to 2-3 rooms while operating indoor.

The unreliability of the wireless communications leads us to explore a novel class of *on-line* algorithms, in which the agents are able to tag the environment and update the state of the tags in order to communicate and coordinate with each other. In particular, we envision the use of inexpensive tags that can be left on the ground and which the agents can use to store and retrieve data (Agent-to-Environment). In our first work [1], we proposed distributed exploration algorithms that are carefully designed to allow agents to interact with the instrumented environment. Agents use smart tags to localize themselves and communicate with other agents on the field. Furthermore, once a network of tags has been deployed, this could be used to provide a much more reliable communication infrastructure between agents and the human personnel outside the building, so that they could know in real-time when the robots find victims or hazards. Another use of the tags could be the storage of information about the area that the first responders could retrieve using PDAs once they enter the building. The feasibility of the approach is supported by Hähnel et al. [19], who proved how a robot can use RFID tags already placed on an area to localize itself and navigate through the rooms, and recently by Kleiner et al.[8], who presented a robot which is able to autonomously drop RFID tags on the environment and implement an existing *on-line* exploration algorithm [20].

One of the first *on-line* methods that uses a cellular decomposition of the environment is the ants-inspired algorithm presented in [7] and [5]. The area is divided into square grid cells on which the exploring agents leave traces of their passage, similarly to real ants leaving pheromone. Agents (or ants) tend to move to the least visited cells, i.e. the cells with the least amount of pheromone, and they increase the number of visits (pheromone) as they hop onto a cell. A similar approach to the Ants algorithm uses a sensor network infrastructure to provide agents with information about the visited areas and direct them to the least recently visited direction [21]. The use of a sensor network is an attempt to cope with the problem of marking the cells in a real scenario, problem which is not really tackled in the original Ants work [7]. In Section 2.2, we describe how we adapted and implemented the original Ants algorithm for our specific research problem detailing strengths and limitations.

A different approach (following an Agent-to-Server communication mode) has been presented in [22]. In this work, the authors use the Boustrophedon technique to cover the area. The environment is divided into cells and each robot starts covering each cell with forward and reverse phases. While doing this, it builds a graph of the environment which is shared by all the robots of the team. Due to this fact, the robots always know where there are uncovered cells and therefore they can distribute themselves over the area in order to cover the remaining unexplored regions. The proposed algorithm is globally simple and easy to implement on real hardware. It also leads toward very efficient coverage time because the robots always know where the unexplored zones are, so they can go directly on them without wasting time in idle states. The graph is shared among all the agents so that, if one of them has a fault, the covering procedure can continue without losing any information. However the assumptions are different from the ones we are adopting for the present work, in particular our agents do not rely on perfect wireless communication among them and we do not assume that they always know where they are in the map (perfect localization).

Yamauchi presents a frontier-based exploration algorithm [23], where the agents explore the environment, represented by a regular grid of cells, keeping in their memory a map of the area and always directing themselves "to the boundary between open space and uncharted territory". A depth first search algorithm is used to move from the current position to the next frontier. Each agent has a local map and a global map shared with all the other agents. When a local map is updated (the agent explores a new area), it is summed with the global map, and the latter is broadcasted to all the other agents so that they can update their global maps. The agents do not broadcast information about what area will be explored next, so different agents could explore the same frontiers in the same time resulting in an inefficient utilization of the team. The algorithm also makes the same strong assumptions that we found in [22] namely perfect localization, communication and mapping, and uses an Agent-to-Server communication approach, because the map is stored in a centralized server.

Another Agent-to-Server communication approach is presented by Howard et al.[24], where the authors show a general approach to explore a building, find objectives, and report them back to the human personnel outside. However, it requires human support to solve problems like loop closures or map merging between the agents, so it does not satisfy the requirements for autonomous area coverage.

Burgard et al. do not assume that the environment is divided into grid cells [25]. Agents compute an utility function to go to the next "frontier" in order to maximize the explored territory. Although the agents have to store information about the map and localize themselves, this approach is probability-based and therefore more suitable for a real scenario: the agents have a list of target points to reach in the next step, each of them associated to a value which takes into account the cost to reach the point, and the probability of exploring new areas once an agent has positioned itself on that point. The probability takes into account how many agents are going to explore the area in which the target point is, therefore avoiding the situation where all the agents explore the same area. Rekleitis et al. [2] try to improve the exploration by mapping the environment while the area is covered by two robots. To localize the robots they use odometry (a position estimate based on the previous movements), but while a robot is exploring the area, the other stands still and observes the former to measure its movements and improve the localization; after a certain amount of time the two robots exchange roles. The authors map the environment as trapezoids divided into striped which are connected forming a graph. The agents use a depth first search algorithm to cover all the stripes. Both the previous approaches can be classified as using an Agent-to-Agent communication. In Section 3.1 of Chapter 3, we propose a variation of the original and well known depth first search algorithm to use multiple agents, namely Multiple Depth First Search algorithm. We describe this algorithm and present its strengths and limitations.

Finally, Batalin et al. [26] focus on agent dispersion and propose two algorithms to make the agents move away from each other when they are in sensing range (Agent-to-Agent communication paradigm). This is an important issue because the agents should always be spread out as much as possible in the environment to avoid wasting resources in exploring the same area multiple times.

2.2 The Ants Algorithm

In this section, we first discuss the behaviour, strengths and limitations of the Ants algorithm proposed by [7, 5]. This is a distributed algorithm that simulates a colony of ants leaving pheromone traces as they move in their environment [5]. Initially, all cells are marked with value 0 to denote that they are *unexplored*. At each step, an agent reads the values of the four cells around it and chooses to step onto the least traversed cell (the one with the minimum value). Before moving there, it updates the value of the current cell,

for example by incrementing its value by one. The authors discuss a few other rules that could be used instead to mark a cell and navigate to the next one, but they all exhibit similar performance in terms of exploration time. Hence, we select the above variant of the Ants algorithm (move to the least visited cell) as a basis for comparison. The authors provide a proof that the agents will eventually cover the entire terrain (provided that it is not disconnected by wall cells), and thus that the *Exploration objective* is always achieved.

The first advantage of the algorithm is its simplicity: agents do not require memory or radio communication, but only one-cell lookahead. Since they are easy to build, many of them can be used to shorten the coverage process. Secondly, there is no map stored inside the agents: if one of them is relocated (accidentally or on purpose) it will not even realise it and it will continue to do its work as if nothing happened. This means that the whole system is flexible and fault tolerant, and the area can be covered even if some markings or agents are lost. At the storage device of each cell, we only need to store an integer counting the number of times that agents have visited the cell. When the number of times exceeds a threshold, the counter is reset to 0.

The main limitation of the Ants algorithm is that the *visited* state is not used to mark the cells, so the *Termination objective* is never achieved, and the agents continue the exploration phase until they run out of energy. Thus, this approach is not suitable in an emergency scenario, in which the primary consideration is to cover the overall area as soon as possible, and be notified immediately after the task is completed. A further drawback of the algorithm is the limited collaboration among agents. As shown in Figure 2.1, in a scenario with many rooms most of the agents would sweep the first few rooms repeatedly, while only a few of them would venture to explore new areas, thus limiting the efficiency of the algorithm when using multiple agents.

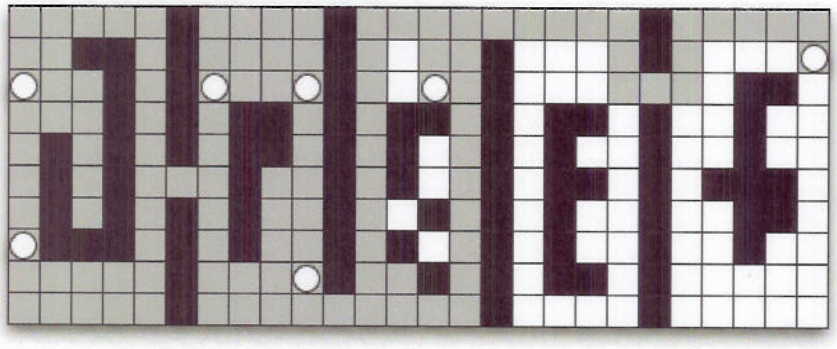

Figure 2.1: The Ants algorithm is not efficient in a scenario with many rooms, because most of the agents explore the first rooms repeatedly, while only few of them set out to discover new areas.

2.3 The Robocup Rescue Competitions

The goal of the urban search and rescue [27] (USAR) robot competitions is to increase awareness of the challenges involved in search and rescue applications, provide objective evaluation of robotic implementations in representative environments, and promote collaboration between researchers. It requires robots to demonstrate their capabilities in mobility, sensory perception, planning, mapping, and *practical operator interfaces*, while searching for simulated victims in unstructured environments.

We carefully checked the approaches of each one of the teams that participated to the most recent edition (2006) and we did not find any autonomous exploration algorithm used by the robots. The aim of the competition is more focused on solving mobility, perception and sensory problems while letting a human operator controlling the robots themselves. A list of all the teams involved in the RoboCup 2006 World Championship is provided below.

Kadous et al. [28] present a preliminary design and implementation of an user interface able to remotely guide a team of heterogeneous, potentially autonomous USAR robots. In Walking Machine [29], the R2K5 robot used for rescue purposes is remotely controlled. In Bremen Rescue Walkers [30], the AIMEE robot [31] is described from a hardware point of view but without any description of autonomous algorithms. In Deutschland1 [32] and in particular in the work [33], the robot is remotely controlled. IUB [34] it is autonomous but there is no exploration algorithm, only victim detection and obstacle avoidance. RescueRobots Freiburg [35], Resko Team [36], Ariana and Aryaak [37], MRL (Naji) [38], Resquake [39], Alcor [40], C-Rescue [41], NIIT BLUE [42], NuTech-R [43], Toin Pelican [44], Roscue [45], RFC Uppsala [46], Independent [47], Good Samaritan [47], RKRS [29] are all projects in which robots are remotely controlled.

Chapter 3

Exploration Algorithms

In this chapter we will present the novel algorithms we devised to let a team of autonomous robots carry on the exploration of an unknown area. Each algorithm will be discussed in details, and it will be compared against the objectives defined in Section 1.2.

3.1 The Multiple Depth First Search Algorithm

In order to address the limitations of the Ants algorithm, we consider a Depth First Search (DFS) approach to traversing the unknown terrain. Unlike Ants, this algorithm allows agents to mark cells as *visited*, so that agents do not need to traverse them in the future. As a result, an agent knows that its task is completed if its four adjacent cells are either *visited* or *wall* cells. The values used to annotate cells by the Ants algorithm are not used in this case.

We first consider the case with a single agent. The agent explores the area by moving to the next *unexplored* cell, marking it as *explored*, and storing in it the direction of the previous cell (i.e., North, East, South, West). In doing so, it builds an exploration tree in which each cell has a parent cell (the cell where the agent came from, before moving to the current cell for the first time). When there are no *unexplored* cells adjacent to the current cell, the agent has reached the end of a branch and is ready to start traversing it backwards. It marks the current cell as *visited* and moves to the parent cell. This is repeated until the agent finds an adjacent *unexplored* cell and moves to it to start marking a new branch as *explored*. In short, most cells (except for leaves cells) are traversed at least twice, once marked as *explored*, as the agent traverses the branch downwards, and once marked as *visited*, as the agent traverses the branch upwards. When the agent is back at the root cell and all adjacent cells are either *visited* or *walls*, the algorithm terminates.

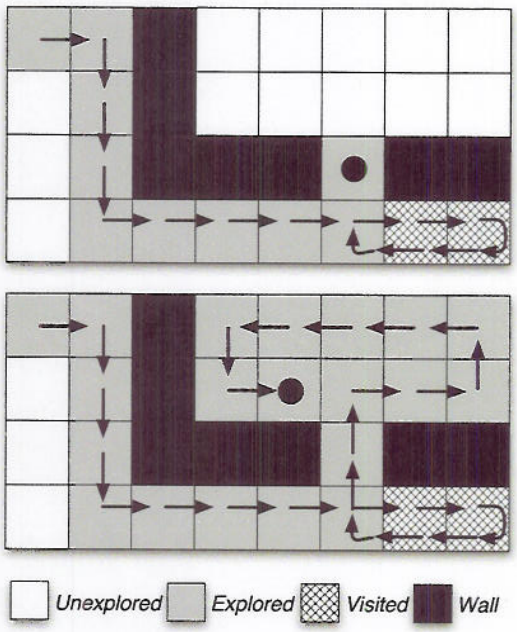

Figure 3.1: Different branches in a Depth First Search exploration.

A snapshot of the exploration process with one agent can be seen in Figure 3.1: the agent starts at the cell in the top left corner of the area, and decides to move on the path denoted by the arrows, annotating cells in the way as *explored*. When it reaches the cell at the bottom right corner, it is surrounded by either *wall* or *explored* cells, and realises that it is at the end of a branch. It starts moving backwards marking the cells of the branch as *visited* until it identifies the start of a new branch. The first cell of the new branch is the one between the two *wall* cells, which is initially *unexplored*. It starts processing the second branch by marking that cell as *explored* and repeating this step as it traverses the branch downwards, until it reaches the current position denoted by the black sphere. At this point, it identifies the end of another branch, and it will start traversing the branch upwards and marking its cells as *visited*. The agent will continue the exploration task in a similar manner until it is surrounded only by *visited* or *wall* cells.

The challenge in using more than one agent is to ensure that they can efficiently collaborate to explore the area. In the extended Multiple Depth First Search (MDFS) algorithm, the protocol that the agents run is a variant of the Depth-First-Search (DFS) one. A detailed description of the MDFS protocol is provided in Algorithm 1.

The main idea is that agents extend the DFS tree with new *explored* cells in their way downwards, and mark these cells as *visited* when they traverse them in the opposite upward direction. In the DFS tree the hierarchical relationship between the cells is defined as parent/child. Moreover, the mechanism used for distributing agents to different parts of the network is as follows: each *explored* cell has a counter that measures how many agents

have traversed it coming from the same parent cell. Note that a cell can be accessed by an agent from at most one parent cell. As agents traverse the DFS tree downwards, they increment the counters associated with each traversed cell. At intersections, they choose to move to the *explored* cells traversed by the minimum number of agents, so as to assign the same number of agents to each branch of the DFS tree.

Using this algorithm the agents are typically able to explore the area in less time than using the Ants algorithm, and more importantly, each agent knows exactly when to stop its exploration task, and thus the algorithm is capable of achieving both the *Exploration* and *Termination objectives*. Hence, the algorithm terminates when all agents stop moving, since when that happens all cells are marked as *visited* or *walls*. Although the Multiple Depth First Search addresses some of the weaknesses of the Ants algorithm, it is still not very efficient in terms of exploration time. By definition it traverses each cell at least twice (except for leaves cells), thus resulting in a long exploration time even in open areas

without walls where a single traversal would suffice.

```
/* The cell in which the agent is located is defined as current cell.
   */
/* The cell from which the agent is coming is defined as previous cell.
   */
/* In the DFS tree the hierarchical relationship between the cells is
   defined as parent/child.                                          */
/* Every parent cell in the DFS tree has a counter for each of its
   children cells.  The counter indicates the number of agents which
   traversed the parent cell toward that particular child cell.      */
```

if *you are coming from a cell which is child of the current one* **then**
  decrease the counter of the current cell associated to the child;
end
if *the current cell is UNEXPLORED* **then**
  mark the current cell as EXPLORED;
  define the previous cell as parent of the current one;
end
if *the parent of the current cell is VISITED AND there is only one child cell AND there are no adjacent UNEXPLORED cells* **then**
  mark the current cell as VISITED;
end
if *there are adjacent UNEXPLORED cells* **then**
  go to one of them (each agent chooses a different cell according to its ID);
end
else if *there is at least one child which is not VISITED* **then**
  go toward the child cell with the minimum associated counter;
end
else
  mark the current cell as VISITED;
  go to the parent cell;
end
if *you are going to a cell which is child of the current cell* **then**
  increase the counter associated to that cell;
end

Algorithm 1: Multiple Depth First Search Algorithm

3.2 The Brick&Mortar Algorithm

Our novel algorithm, named Brick&Mortar, is designed to address the weaknesses of the algorithms seen so far. Unlike the Ants algorithm, agents using Brick&Mortar know when the exploration task is completed and they do not spend much time revisiting the same cells. Unlike MDFS, they typically traverse each cell less than twice, thus resulting in a shorter exploration time.

The driving idea is that of thickening the existing walls by progressively marking the cells that surround them as *visited* (see Figure 3.2). Once again, *visited* cells are equivalent to *wall* cells in that they can no longer be accessed. In the description of the algorithm, we refer to *wall* and *visited* cells as inaccessible cells, and to *unexplored* or *explored* cells as accessible cells. The algorithm aims to progressively thicken the blocks of inaccessible cells, whilst always keeping accessible cells connected. The latter can be achieved by maintaining corridors of *explored* cells that connect all *unexplored* parts of the network as shown in Figure 3.2.

Like Ants and MDFS, Brick&Mortar does not require agents to know their location in the building. A relocated agent can simply navigate randomly until it finds an accessible cell and then continues the exploration from there. Brick&Mortar makes the blocks of inaccessible cells thicker until the entire terrain is converted to a large block of inaccessible cells. In a rectangular terrain without *wall* cells, agents starting from border cells always succeed in visiting the entire area. In more complex topologies with many rooms and obstacles, agents may be faced with a loop closure problem described below. We are now in a position to introduce the details of the proposed Brick&Mortar algorithm with and without loop closure.

3.2.1 Brick&Mortar Without Loop Closure

The Brick&Mortar algorithm (see Algorithm 2) consists of two discrete steps. In the *marking step*, the agent marks the current cell choosing between the *explored* and *visited* states. In the *navigation step*, the agent decides which cell to go to next.

Marking step: Every time an agent is in an *unexplored* cell (with no tags on it), it deploys a tag and updates the state of the cell, choosing between the *explored* and *visited* states. The current cell is marked as *visited* if it does not block the path between two accessible cells around it. Otherwise it is marked as *explored*. Figure 3.3 provides two examples of the marking step: one where the current cell is marked as *explored* (map a), and one where it is marked as *visited* (map b). In the first example, the only path between the two *unexplored* cells A and B traverses the cell in which the agent (black spot) is at the moment, thus the cell cannot be marked as *visited*. In the second example, there is an

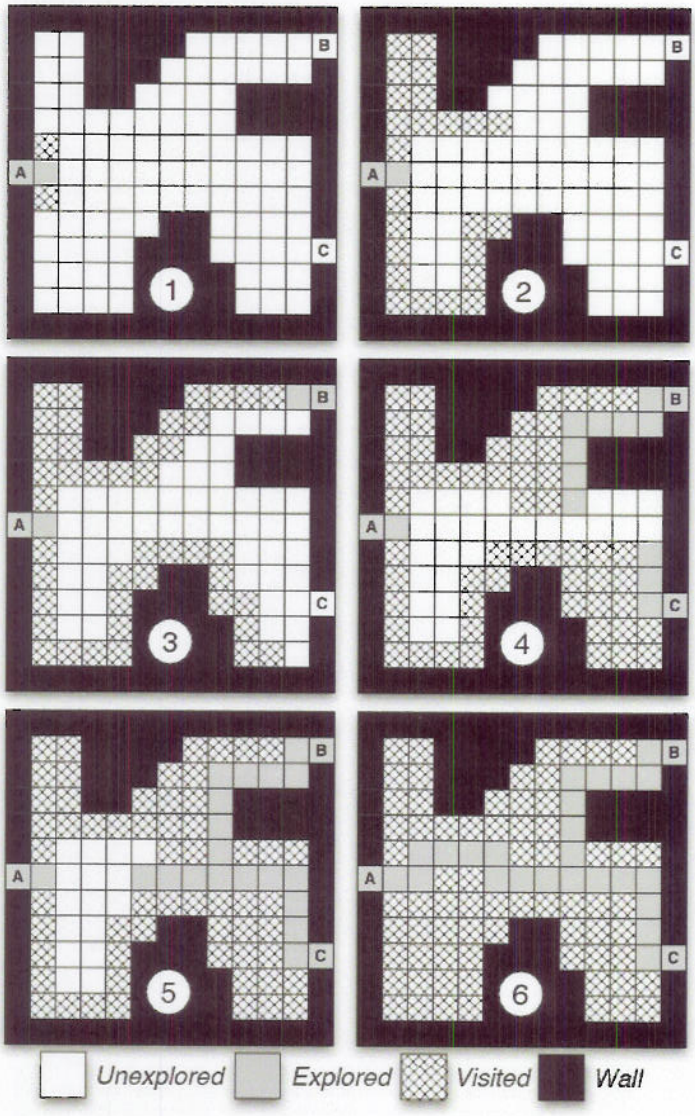

Figure 3.2: Two physical agents running Brick&Mortar are used to explore a room, by gradually deploying tags in its cells and updating their states. Cells A, B and C represent doors. The agents enter the room from door A and gradually thicken the walls by marking cells adjacent to walls as *visited* (Stages 1, 2 and 3). Recall that visited cells cannot be accessed in the future by other agents - thus, they can be viewed as virtual walls. Agents stop thickening walls if they are at risk of disconnecting two unexplored parts of the network. For example, in Stage 4, the physical agents create two corridors of *explored* cells in order not to disconnect *unexplored* cells in the inner part of the room from unexplored cells in other rooms (beyond doors B and C). When the exploration of the current room is finished (Stage 6), the cells A, B and C are connected by corridors of *explored* cells. These corridors will allow agents to traverse the room through doors A, B and C to access other unexplored rooms in the building. These corridors will be eventually marked as *visited* when no room in the building is left unexplored.

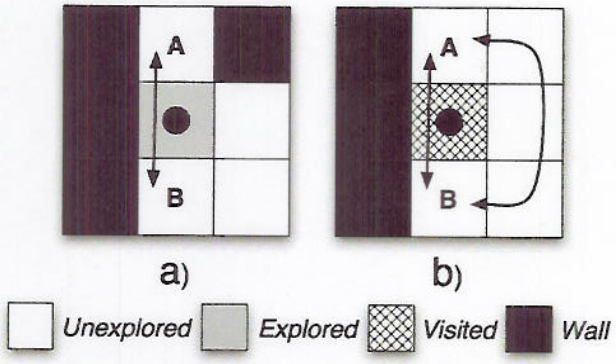

Figure 3.3: Marking rules: the agent must decide to mark the current cell as *explored* or *visited*. In the first example (a) the current cell cannot be marked as *visited* because it is the only available passage between adjacent cells A and B. In the second example (b) the current cell is marked as *visited* because there is an alternative passage between A and B on the right.

alternative path on the right end side of the map, thus the current cell can be marked as *visited* without closing the way between A and B. Note that such alternative paths are easy to compute locally, because they are strictly confined to the 8-cell perimeter of the current cell.

Navigation step: In this step, the agents take a decision about which cell to access next. Priority is always given to the *unexplored* cells which are adjacent to the current one (Figure 3.4b). If the *unexplored* cells are more than one, the cell which is most likely to be marked as *visited* is chosen, i.e. the one with the most black cells around it (Figure 3.4c). If many adjacent cells are *unexplored* and no one is more likely to be marked as *visited*, one of them is chosen at random (Figure 3.4d). If there are no *unexplored* cells, an *explored* cell is selected according to the ID of the agent (so that different agents spread in different direction). Finally, when the agent is surrounded by inaccessible (*wall* or *visited*) cells, it stops its exploration task (Figure 3.4a).

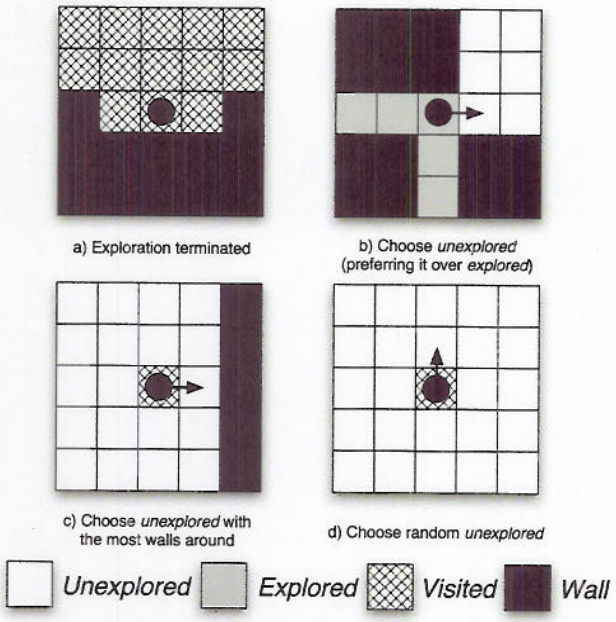

Figure 3.4: Different situations in which the agent applies the navigation rules to decide which one of the adjacent cells it will go to during the next move.

Details of the two steps are provided in Algorithm 2:

```
   /* Marking step                                                    */
 1 increase the counter of the cell;
 2 if the cell is not blocking the path between two other cells then
 3 |  mark it as VISITED;
 4 end
 5 else if the cell is UNEXPLORED then
 6 |  mark it as EXPLORED;
 7 end
   /* Navigation step                                                 */
 8 if one of the adjacent cells is UNEXPLORED then
 9 |  go to the one with the most walls around it;
10 end
11 else if at least one of the adjacent cells is EXPLORED (and different from the cell you
   are coming from) then
12 |  go to one of them selecting the first EXPLORED cell in an ordered list of adjacent
   |  cells;
   |  /* the order of cells in the list depends on the agentID, so that
   |     different agents disperse in different directions            */
13 end
14 else
15 |  stay in the current cell;
16 end
```

Algorithm 2: Brick&Mortar Algorithm

3.2.2 Brick&Mortar With Loop Closure

The simple version of Brick&Mortar, described in the previous subsection, terminates successfully (achieving both the *Exploration* and *Termination objectivs* only if agents do not encounter loops during the exploration process. Informally, a loop occurs when an agent traverses the same sequence of *explored* cells multiple times without being able to mark any of the cells as *visited*. Loops are encountered when there are clusters of *wall* cells in the middle of an area. For example, in Figure 3.5a, an agent on cell C_1 of the figure will start building a corridor of *explored* cells traversing cell C_2 and then finding itself back at cell C_1 again. According to the rule in the marking step of Algorithm 2, every cell blocks the path between the previous and the following one, and is thus repeatedly marked as *explored*. The loop problem is well known in the literature, but usually the proposed algorithms either ignore it [3] or need an external human operator to solve it [24]. In emergency scenarios, the team of agents might be inside a building or underground far away from the rescue team, and it should still be able to accomplish its exploration mission. For this reason we extend the original version of Brick&Mortar algorithm to enable loop closure without human intervention.

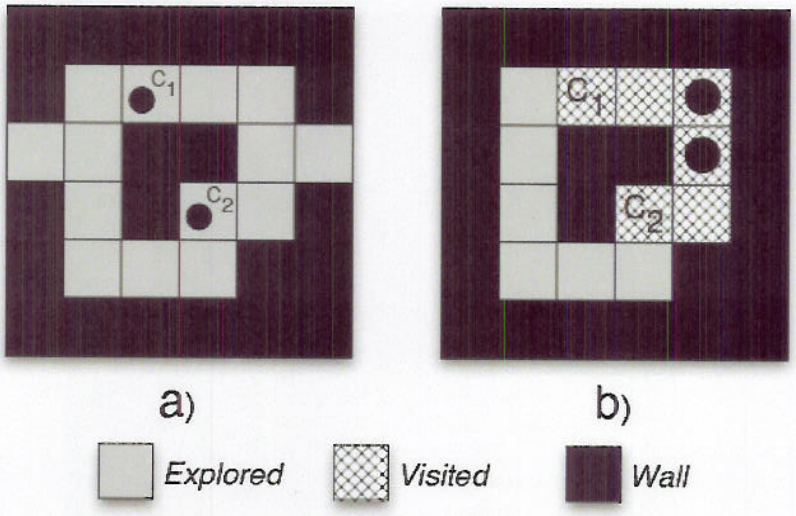

Figure 3.5: The loop problem.

To make the algorithm capable of closing loops, and achieve the *Exploration* and *Termination objectivs* in every situation, we make an additional assumption: an agent is able to mark a cell with its ID (a simple number identifying the agent) and the directions (North, East, South or West) in which the agent is moving in and out of the cell. This assumption is not too strong because each agent can be equipped with a small and inexpensive

electronic compass that detects and controls the direction of its movement. An agent detects the presence of a loop when it traverses the same *explored* cell twice in the same direction. If only one agent is performing the exploration task, it can easily break the loop by marking one of the cells as *visited* and then resuming its original wall-extension strategy (i.e. the marking and navigation steps described in Section 3.2.1).

However, if the terrain is explored by multiple agents, independent attempts to close the same or overlapping loops may result in agents being trapped within inaccessible areas and being unable to help with the remaining exploration task. For example, two agents A_1 and A_2 can traverse the loop of Figure 3.5a once starting from cells C_1 and C_2 respectively and moving in opposite directions. As they traverse the loop, they mark each cell as *explored*. Once agents A_1 and A_2 reach cells C_1 and C_2 respectively, they detect the presence of a loop. To resolve the loop, they mark C_1 and C_2 as *visited*, and continue to move towards each other marking all cells in their way as *visited* (Figure 3.5b). Once they meet, they get trapped because they are surrounded by inaccessible (*visited* or *wall*) cells without having succeeded in marking the entire loop as *visited*.

In order to avoid being trapped, loop resolution is performed in four phases, shown in Figure 3.6:

- **Loop detection:** Initially, an agent follows the marking and navigation steps described in Section 3.2.1 leaving a trace in each cell that it traverses (how it moved out of it), until it detects a loop. This happens when it moves into the same *explored* cell a second time (but not in the opposite direction than the one used previously to move out of that cell). Upon detecting a loop, the agent moves to the loop control phase. This phase requires $O(N)$ storage capacity at each cell, where N is the number of agents.

- **Loop control:** To take control of the loop, an agent A starts traversing the loop a second time in the same direction, trying to take control of each cell by annotating it with its own agent ID_A (this requires $logN$ bits, where N is the number of agents). It is possible only if the cell is explored and it is not already annotated with another agent's ID (say ID_B). Agent A knows that it has succeeded in taking control of all cells in the loop when it steps again on a cell that is already annotated with the same ID. In this case, it immediately switches to the loop closing phase.

- **Loop closing:** The agent is now in a position to break the loop by marking the current cell as *visited*, and continuing to do so until it reaches the first intersection (the cell with at least one *explored* neighbor cell that does not belong in the loop). The agent then switches to the loop cleaning phase.

- **Loop cleaning:** The agent removes any traces of the loop control phase, by moving backwards in the loop, removing its ID from the cells that it previously annotated, and resetting the direction in which the agent moved out of the cells to null.

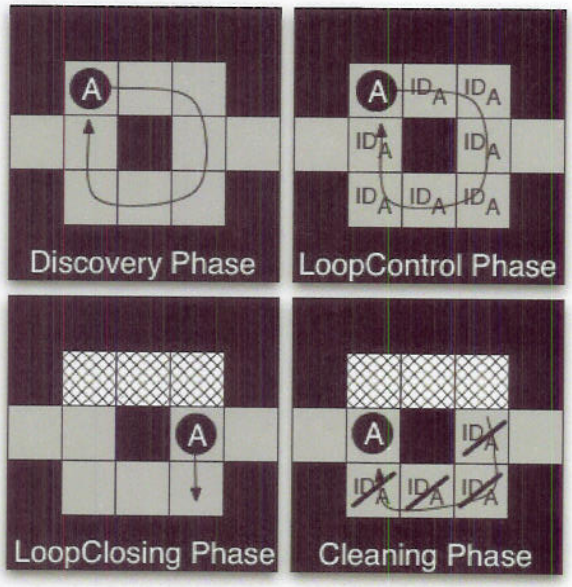

Figure 3.6: Phases of the loop resolution mechanism.

It remains to describe in detail what happens in the loop control phase when an agent A is not able to take control of a cell, either because the cell is already *visited*, or because it is being controlled by another agent B. To deal with this scenario, we allow agents to interrupt the loop control phase, either permanently or temporarily. In the former case, an agent literally quits the loop resolution mechanism and moves to the loop cleaning phase. In the latter case, the agent can wait at a cell, as a *standby* agent, until the state of the cell is changed.

In what follows, we provide specific rules that define how an agent decides whether to continue, quit or stall controlling cells with its ID during the loop control phase. Let agent A try to move to the next cell in order to control it (i.e., to annotate it with ID_A). The decision it makes depends on the state in which it finds the cell:

$$\begin{cases} \textit{control cell}, & \textit{if cell explored without control} \\ \textit{start loop cleaning}, & \textit{if cell visited} \\ \textit{start loop cleaning}, & \textit{if } B \textit{ controls } \& \ (ID_B > ID_A) \\ \textit{start loop cleaning}, & \textit{if } C \textit{ standby } \& \ (ID_C > ID_A) \\ \textit{become standby}, & \textit{otherwise} \end{cases}$$

We finally need to define how, a standby agent (which waits at a cell as a result of following the last branch above), reacts to changes in the cells state. The rules that determine its

behavior are:

$$\begin{cases} \textit{start loop cleaning}, & \textit{if replaced by another agent} \\ \textit{start loop cleaning}, & \textit{if cell becomes visited} \\ \textit{continue loop control}, & \textit{if cell is cleaned} \\ \textit{remain standby}, & \textit{otherwise} \end{cases}$$

An agent that manages to complete its loop control phase has succeeded in controlling *all common cells with other interfering loops*. It performs loop closing, and then cleans all common cells, before another interfering agent gets the opportunity to take control of them. Hence, although in general loops are handled concurrently, agents are able to detect when their loops interfere, and in this case, they resolve them in a sequential manner. The loop control phase has a similar role as locks in database systems, i.e. it allows concurrent operations whilst leaving the system in a *consistent* state. In our case, consistency means that all *explored* and *unexplored* cells remain connected, and no agent is trapped within *visited* cells.

Using the rules above we can prove that when agents try to resolve overlapping loops, they never cause a deadlock, they never get trapped within *visited* areas, and they always terminate. Both the *Exploration* and *Termination objectivs* are therefore achieved by the Brick&Mortar algorithm in its version with loop closure. However, to ensure this we require memory capacity at each cell that grows linearly in the number of agents (similar to MDFS storage requirements).

3.3 The HybridExploration Algorithm

In this section, we introduce another novel algorithm, called the HybridExploration algorithm, able to gracefully combine two parallel protocols, one followed by physical agents (robots, or simply agents) and one followed by virtual agents. A physical agent takes significantly longer to move from one cell to another (physical robot motion) than a virtual agent (message propagation). Hence, the time of completing the exploration task is measured as the minimum number of physical steps required by physical agents to explore the entire area. Furthermore, another novelty is in the use of the Ants algorithm to modify the original Brick&Mortar approach and let the physical agents (which are running it) to achieve the *exploration objective*.

3.3.1 Physical Agent Protocol

The physical agent protocol is a modified version of the Brick&Mortar algorithm without loop closure, previously described in Section 3.2.1. To make the agents capable of

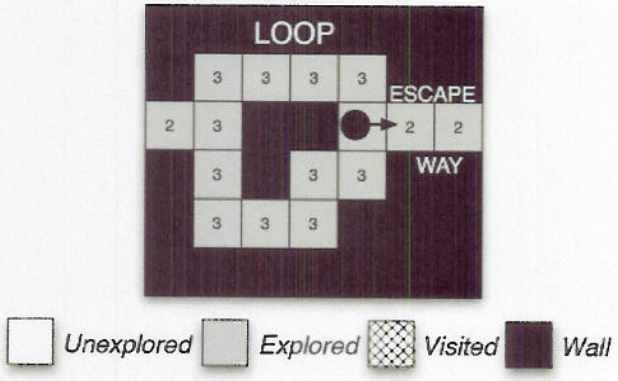

Figure 3.7: The numbers in the cells represent the times the cell has been traversed in the past. The agent goes to the cell which has been traversed the minimum amount of times to avoid being trapped in a loop.

achieving the *Exploration Objective*, we introduced a mechanism which is similar to the Ants algorithm, and let the agents escape from the loops of *explored cells* in which they might be trapped. To do that, we need to introduce a counter for each tag, which can be read or modified by the agents. Once we have this, the agents simply need to increase a counter on the tag of a cell each time they traverse it, and then read all the counters of the adjacent cells and choose the minimum. In this way, an agent which traverses the same cell twice can take different decisions about the next move instead of always going toward the same direction, thus avoiding being trapped in a loop like the one shown in Figure 3.7.

The protocol enables them to cover the entire area of interest eventually, but it has two weaknesses. First, physical agents are often inefficient in exploring the area, as they cover the same cells multiple times, instead of focusing their efforts on unexplored parts of the space. Second, although physical agents eventually manage to visit every cell at least once, they are not aware when this happens, i.e. they have no indication of when the exploration task terminates. These two problems are discussed in detail in Section 3.3.2 and they motivate the introduction of the *Virtual Agent Protocol* in Section 3.3.3. The two protocols, the Physical Agent Protocol and the Virtual Agent Protocol, work in synchrony and together constitute our proposed HybridExploration algorithm.

3.3.2 Analysis of the Physical Agent Protocol

In this section, we highlight the strengths and weaknesses of the Physical Agent Protocol. The weaknesses motivate the need to extend it with a new protocol, the Virtual Agent Protocol, which we describe in detail in Section 3.3. We assess the behaviour of the Physical Agent Protocol with regards to the objectives specified in Section sec:objectives.

Advantage 1. *The Physical Agent Protocol always achieves the* Exploration Objective.

Proof. Let's define a cell as *accessible* if the cell is *explored* or *unexplored*. Because of the marking rules illustrated in Figure 3.3, *accessible* cells always remain part of the same group, i.e. for each pair of *accessible* cells A and B an agent is always able to go from A to B (and vice versa) only traversing *accessible* cells. Agents always move towards an adjacent *accessible* cell with the smallest counter value (we can consider *unexplored* cells as *accessible* cells with a counter $= 0$) and, once there, they increase the local counter value. According to the proof in [7], this guarantees that agents will eventually traverse all *unexplored* cells (with counter 0) and mark them as *explored* or *visited*. □

The performance of the Physical Agent Protocol is severely compromised in areas with physical obstacles, like desks and walls in the middle of rooms. Let's first define the term *obstacle* formally, and then discuss the weaknesses of the Physical Agent Protocol in the presence of obstacles.

First of all, we need to specify that two cells are linked if one is in the 8-cell perimeter of the other. Thus, a cell can have up to 8 linked cells (in the North, East, South, West, North-East, North-West, South-East, South-West directions), whereas only up to 4 adjacent cells (in the North, East, South or West directions). We can now define an obstacle O as a set of cells with the following properties:

- Each cell $c \in O$ is inaccessible, i.e. *wall* or *visited*.
- Any cell c' linked to a cell $c \in O$ belongs to the same obstacle ($c' \in O$).
- Each pair of cells in obstacle O is connected via cells of the same obstacle. That is, for any pair of cells c_1 and c_n in obstacle O, there is a sequence of cells $c_1, \ldots, c_n$, such that $c_i \in O$ and c_{i+1} is linked to c_i, for all $i = 1, \ldots, n-1$.
- None of the obstacle cells is linked to a cell in the perimeter of the map.

Intuitively, we can think of an obstacle as an *island* of inaccessible cells separated from the perimeter of the map by other accessible cells.

Disadvantage 1. *If there are obstacles in the area, the Physical Agent Protocol can never achieve the* Termination Objective.

Proof. Initially, there are many cyclic paths of accessible (*unexplored* or *explored*) cells around each obstacle (see cyclic paths of white cells that wrap around the obstacle on the left map of Figure 3.8). As agents mark more cells as *visited* the number of such paths gradually decreases (circular paths of white cells that wrap around the obstacle on the right map of Figure 3.8). Assume that the Physical Agent Protocol was about

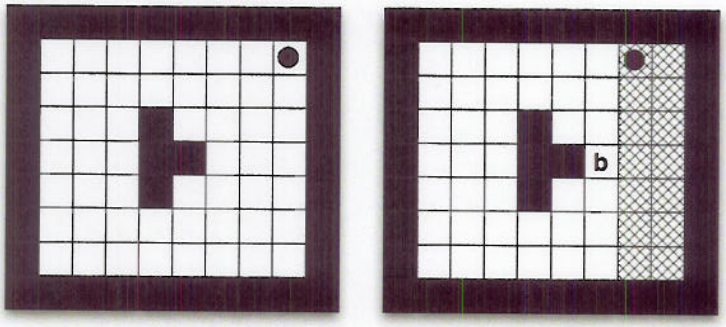

Figure 3.8: The number of cyclic paths of accessible (*unexplored* or *explored*) cells around an obstacle gradually decreases as the agent marks more cells as *visited*. For example, the right map has fewer cyclic paths of white cells around the black cells than the left path. However, it is not possible to remove all cyclic paths by marking one of their common cells (e.g. cell b) as visited, as this would violate the marking rules of the Physical Agent Protocol, illustrated in Figure 3.3.

to remove the last set of remaining cyclic paths, by marking a single cell as *visited*. For this to happen, this cell should be common to all remaining cyclic paths, like cell b on the right map of Figure 3.8. Since this cell belongs to cyclic paths of accessible cells, by removing it, we would disconnect the accessible cells that are adjacent to it on the cyclic paths. This however is not possible according to the marking rules of the Physical Agent Protocol illustrated in Figure 3.3. Hence, it is impossible to remove all cyclic paths of accessible cells around an obstacle, which means that it is impossible to achieve the Termination Objective in the presence of obstacles. An illustrative step-by-step example of this problem in a specific map scenario is provided in Figure 3.9. □

Disadvantage 2. *If there are obstacles in the area, the Physical Agent Protocol is considerably slowed down in achieving the* Exploration Objective.

An example of the second disadvantage is shown in Figure 3.10, where an agent is exploring the same loop several times, thus wasting time that could be employed in exploring unknown areas of the map. The cells in the loop (part L of the map) have been traversed only once. To go from the zone L toward the unexplored part of the map (U), the agent in the loop will need to explore every cell in it at least other 3 times, so that once traversing the cell pointed by the arrow, it will be able to choose the corridor on the left which leads toward the zone U. Thus, before being able to escape the loop, the agent will need to traverse $3n$ cells, where n is the number of cells in the loop. The bigger the loop, the more time the agent will spend in it, without being able to collaborate with the other agents in exploring other areas of the map. This waste of resources (agents) causes a longer overall exploration time, which is an essential parameter to be minimized in an emergency scenario.

To summarize, we have shown that whereas the Physical Agent Protocol always achieves

the exploration objective, it takes considerably longer to do so in the presence of obstacles. In addition, it can never achieve the termination objective in the presence of obstacles. Both problems arise from the inability of the Physical Agent Protocol to cope with loops of accessible cells formed around obstacles. In the next section, we introduce a new protocol, called the Virtual Agent Protocol, which provides a loop-resolution mechanism, and works in synergy with the Physical Agent Protocol. Together the two protocols constitute the HybridExploration protocol that is capable of achieving both exploration and termination objectives in an efficient manner.

3.3.3 Virtual Agent Protocol

To speed up the exploration process and eventually achieve the termination objective, we introduce the concept of virtual agents. These are active messages propagated from cell to cell via the corresponding wireless tags. Virtual agents can only move to cells that are already deployed with tags; they cause changes in the current cell's state and make informed decisions about which cell to traverse next. Like physical agents, virtual agents move from one cell to an adjacent cell in the north, east, south or west direction, and they cannot traverse *visited* or *wall* cells. Unlike physical agents, they cannot traverse *unexplored* cells since there are no tags deployed there. Hence, they consider *explored* cells as their own territory, and build DFS trees along the corridors of *explored* cells that the physical agents leave behind. The goal of the virtual agents is to remove cyclic paths (loops) of *explored* cells.

The protocol that they run is a variant of the Depth-First-Search (DFS) algorithm. A detailed description of the Virtual Agent Protocol is provided in Algorithm 3. The main idea is that virtual agents extend the DFS tree with new *explored* cells in their way downwards, and mark these cells as *visited* when they traverse them in the opposite upward direction (as shown in lines 8 – 14 and 15 – 28 of Algorithm 3). An illustrative example of the Virtual Agent Protocol is provided in Figure 3.11. The two agents first move together and include each *explored* cell in their way into the DFS tree (as shown in lines 4 – 7 of Algorithm 3). At the intersection, they continue to extend the DFS tree, but the one extends the left branch and the other the right branch. When the two virtual agents meet, they cannot extend the DFS tree any further; hence, they traverse the branches upwards marking cells in the way as *visited*.

The example above showed that the Virtual Agent Protocol tries to balance its resources across different branches of the DFS tree (one virtual agent followed the left branch and the other the right branch). The mechanism used for distributing virtual agents to different parts of the network is as follows: Each *explored* cell has a counter that measures how many virtual agents have traversed it coming from the same parent cell. Note that a cell can be accessed by a virtual agent from at most one parent cell. As virtual agents traverse

the DFS tree downwards, they increment the counters associated with each traversed cell (as shown in lines 29 – 31 of Algorithm 3). At intersections, they choose to move to the *explored* cells traversed by the minimum number of virtual agents, so as to assign the same number of agents to each branch of the DFS tree. In Figure 3.12, the second virtual agent will go right, because the counter associated to that direction is 1 at the moment, whereas the counter associated with the left direction is 2.

We now discuss two rules that virtual agents should follow to work in harmony with physical agents. These rules are handled in lines 9 – 12 and 22 – 25 of Algorithm 3.

Rule 1: a virtual agent cannot mark a cell as *visited* if that cell is occupied by a physical agent, and it always has to wait until the physical agent is gone before continuing marking. An example of this behaviour is depicted in Figure 3.13(A), where virtual agents are following physical agents while they avoid "overtaking" them and subsequently blocking them in a branch with *visited* cells.

Rule 2: a virtual agent cannot mark a cell as *visited* if at least one of the adjacent cells is *unexplored*. An example of this case is provided in Figure 3.13(B), where the virtual agent stops any activity until the adjacent cell is explored by a physical agent.

To summarize, the Virtual Agent Protocol is capable of removing cyclic paths of *explored* cells, formed by the Physical Agent Protocol. The role of virtual agents is to assist physical agents, by following them closely and cleaning up unwanted *explored* states in some of the cells, in order to help physical agents achieve the exploration and termination objectives faster. The rules followed by virtual agents ensure that they always work in harmony with physical agents, i.e. they never delay or trap physical agents before the termination objective is achieved. For a detailed description of the Virtual Agent Protocol,

the interested reader can refer to the pseudocode provided in Algorithm 3.

```
/* The cell in which the agent is located is defined as the current cell.
                                                                        */
/* The cell from which the agent is coming is defined as the previous
   cell.                                                                */
/* In the DFS tree the hierarchical relationship between the cells is
   defined as parent/child.                                             */
/* Every parent cell in the DFS tree has a counter for each of its
   children cells.  The counter indicates the number of virtual agents
   which traversed the parent cell toward that particular child cell.
   */
1  if you are coming from a cell which is child of the current one then
2  |  decrease the counter of the current cell associated with the child;
3  end
4  if the current cell is not part of the DFS tree then
5  |  mark the current cell as part of the DFS tree;
6  |  define the previous cell as parent of the current one;
7  end
8  if the parent of the current cell is VISITED AND the current cell has one child cell only
   AND all the adjacent EXPLORED cells are part of the DFS tree then
9  |  if there are any adjacent UNEXPLORED cells OR a physical agent is in the current
   |  cell then
10 |  |  stay in the current cell;
11 |  |  return
12 |  end
13 |  mark the current cell as VISITED;
14 end
15 if there are EXPLORED adjacent cells which are not part of the DFS tree then
16 |  go to one of them (each agent chooses a different cell according to its ID);
17 end
18 else if there is at least one child which is not VISITED then
19 |  go toward the child cell with the minimum associated counter;
20 end
21 else
22 |  if there are any adjacent UNEXPLORED cells OR a physical agent is in the current
   |  cell then
23 |  |  stay in the current cell;
24 |  |  return
25 |  end
26 |  mark the current cell as VISITED;
27 |  go to the parent cell;
28 end
29 if you are going to a cell which is child of the current cell then
30 |  increase the counter associated to that cell;
```

3.3.4 Analysis of the HybridExploration Algorithm

Theorem 1. *The HybridExploration algorithm, which consists of the Physical Agent Protocol and the Virtual Agent Protocol, always achieves the* Exploration Objective.

Proof. In Section 3.3.2, we proved that the Physical Agent Protocol alone always achieves the exploration objective, i.e. the physical agents eventually leave no cell in the *unexplored* state. The rules illustrated in Figure 3.13 ensure that virtual agents do not block physical agents from achieving the exploration objective. The reasons are that virtual agents never disconnect two accessible parts of the network, and they never trap physical agents within *visited* cells away from accessible cells. □

Theorem 2. *The HybridExploration algorithm always achieves the* Termination Objective.

Proof. When the exploration objective is achieved, all *explored* cells in the network are connected, since none of the Physical or Virtual Agent Protocols ever disconnect accessible parts of the network. Hence, the virtual agents will eventually finish building a single DFS tree over these *explored* cells and will eventually mark them as *visited* when traversing the branches of the tree upwards. □

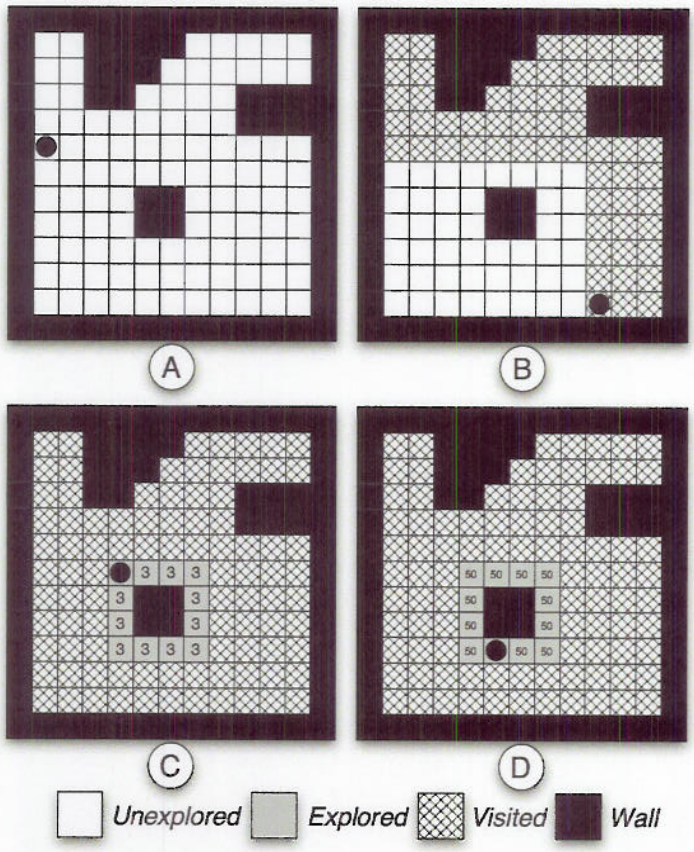

Figure 3.9: A physical agent explores a room with an obstacle in the middle of it. At the beginning, all the cells in the room are *unexplored* (Stage A). While the agent explores the room, it thickens the walls with layers of *visited* cells (Stage B) until it forms a loop of *explored* cells around the obstacle (Stage C). To close the loop, the agent needs to mark one of the cells in it as *visited*. However, each cell is blocking the path between the two cells adjacent to it, and although an alternative path exists (the loop itself), the agent cannot know it locally. Therefore, no cell in the loop is marked as *visited*, and although the room has been fully explored, the agent will just keep moving along the corridor forever (Stages C and D).

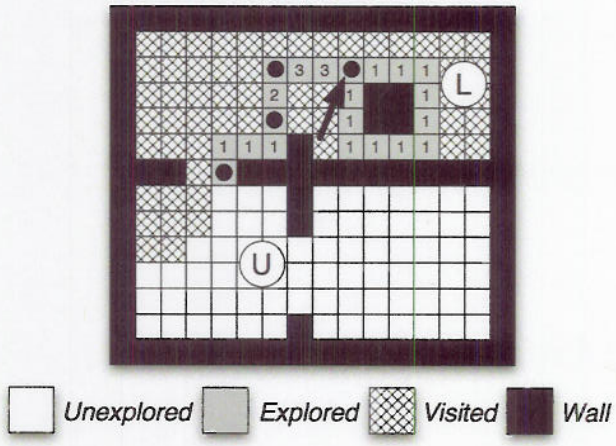

Figure 3.10: The cells in the loop (part L of the map) have been traversed only once. To go from the zone L toward the unexplored part of the map (U), the agent in the loop will need to explore every cell in it at least other 3 times, so that once traversing the cell pointed by the arrow, it will be able to choose the corridor on the left which leads toward the zone U. This waste of resources (agents) causes a longer overall exploration time, which is an essential parameter to be minimized in an emergency scenario.

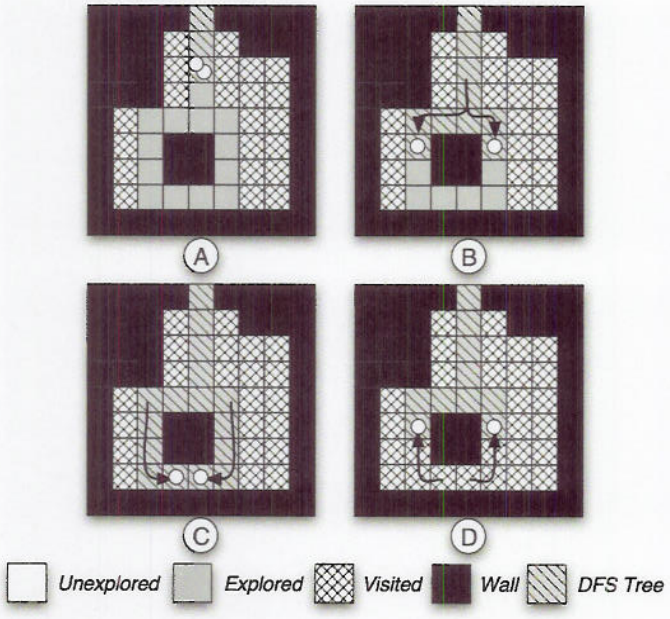

Figure 3.11: Two virtual agents are collaborating to close a loop left by the physical agents. After the start (A), they separate into each of the two branches of the DFS tree (B) and meet again at the other side of the loop (C). Once there, they do not find any more *explored* cells which are not part of the DFS tree, so they close the loop by going back and marking every cell as *visited* (D).

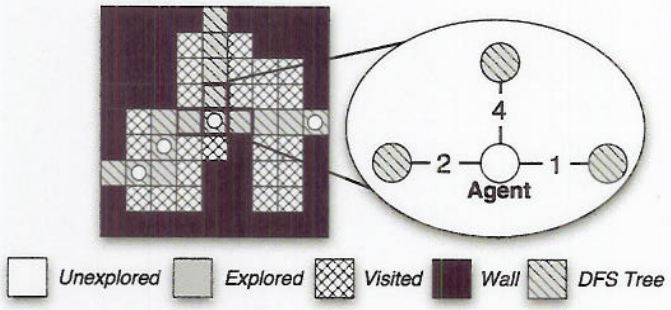

Figure 3.12: In the DFS tree, each cell has four counters, associated with every edge connecting the current cell to the adjacent ones. The counter is updated by the agents and represents the number of agents which traversed the cell and went in the direction of that particular edge. The aim of the counters is to balance the agents during the exploration of the tree: at each node in fact, an agent will choose the edge which has been explored by the minimum number of other agents, so as to assign the same number of agents to each branch of the DFS tree. In the figure, the second virtual agent will go right, because the counter associated to that direction is 1 at the moment, and the previous agent has already updated the counter of the edge on the left from 1 to 2.

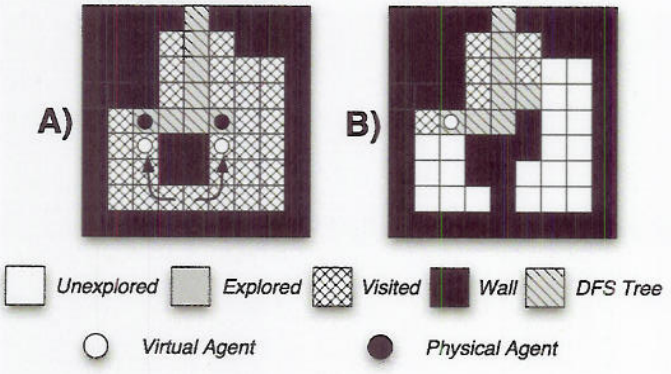

Figure 3.13: Exploration rules for virtual agents. (A) Since the virtual agents are just messages, they move infinitely faster than the physical ones. To avoid virtual agents marking as *visited* a branch of the DFS tree in which a physical agent is exploring, the virtual agents always stop when they need to mark as *visited* a cell which is occupied by a physical one. As a result, a virtual agent could follow a physical one and mark the cells immediately behind it, literally "pushing" the physical agent toward unexplored areas. (B) When a virtual agent needs to mark as *visited* a cell with at least an *unexplored* adjacent neighbour, it stops until the unknown area is explored by a physical agent. In this way, unexplored areas will never be disconnected from the explored corridors.

Chapter 4

Evaluation

This chapter will present the experimental data that we gathered from running the proposed algorithms in both simulated and real environments. The results will be discussed in details, and all the algorithms will be compared against each other to analyse their strengths and limitations.

4.1 Simulation Environment

We developed a simulation tool to test the performance of all the proposed algorithms (HybridExploration, MDFS [1] and Brick&Mortar [1]). In fact, MDFS and Brick&Mortar are improved versions of the ones presented in [1]: MDFS agents are now capable of coordinating so that they are not trapped among *visited* cells; the loop resolution protocol of Brick&Mortar has been improved using timestamps and now is faster than the original version proposed in [1]. To introduce a further comparison with the existing literature, we also introduced the results achieved by the Ants algorithm presented by Svennebring et al. in [5]. The Ants algorithm performance appears in the graphs reporting only the exploration times, because this approach is not capable of achieving the termination objective. This happens because the Ants agents are not able to identify when the exploration terminates, therefore all cells remain permanently marked as *explored* and none of them is subsequently marked as *visited*. A detailed comparison between Ants and the original Brick&Mortar algorithm can be found in [1]. The developed tool allows us to automatically generate terrain maps with different topological features. Thus, we are able to study the impact of i) the number of obstacles, ii) the terrain size, and iii) the number of rooms on the performance of the three algorithms. Each point in the following graphs is the average of running an algorithm 20 times, each time with a different randomly generated map that satisfies the input topological features. In each experiment, we vary the values of one parameter, and assign default values to the remaining ones. The default values

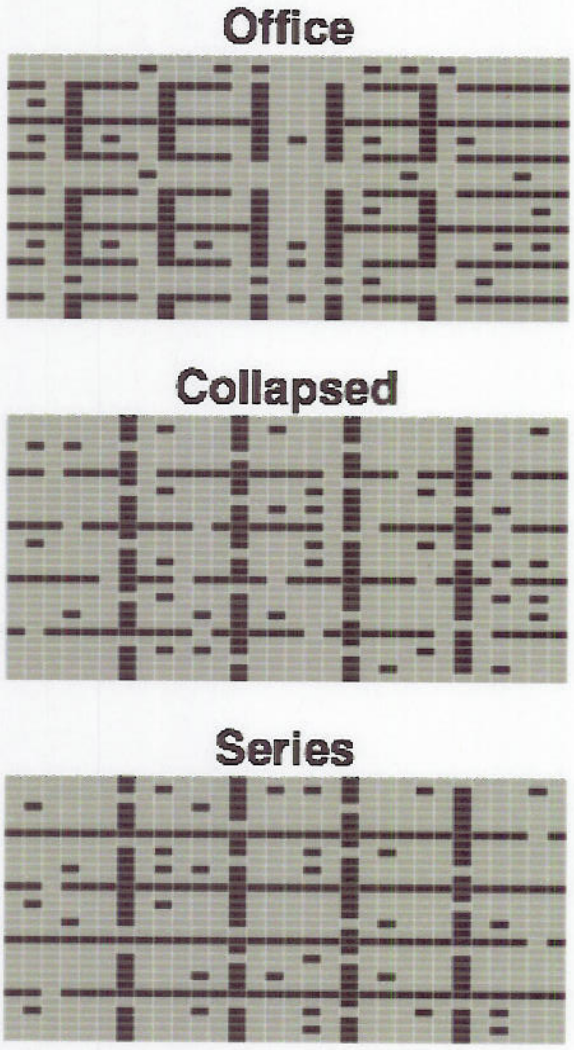

Figure 4.1: Different scenarios in which the algorithms were tested.

are: a map of 2500 (50x50) cells with 30 obstacles and 36 (6x6) rooms, which is explored by 20 agents. The agents are deployed from the top left cell of the area. We consider two performance metrics: i) the *exploration time* (each graph on the left of the following figures), i.e. the number of steps required to achieve the exploration objective, and ii) the *visiting time* (each graph on the right of the following figures), i.e. the number of steps required to achieve the termination objective.

We also consider three different area types: i) *Office*: area inspired by a real building plan, with corridors, offices, and an open-space area. ii) *Collapsed*: area in which a severe event occurred (i.e., earthquake) and disrupted the normal plan of the building. iii) *Series*: a long corridor which traverses several rooms; this scenario is inspired by a mine or another underground map in which each room has a door entering from the previous

one and another door leading to the next. Examples of these area types are provided in Figure 4.1. Our objective is to investigate how the performance of the proposed and competing algorithms varies depending on the spatial layout of rooms and doors in a building.

4.2 Simulation Results

Effect of obstacles: Let us first study the impact of obstacles on the performance of HybridExploration and competing algorithms, as shown in Figure 4.2. In all three scenarios, the introduction of obstacles does not seem to slow down MDFS towards achieving the exploration and termination objectives. In contrast, Brick&Mortar, which has a complex and time-consuming mechanism for resolving loops around obstacles [1], suffers from having to resolve an increasing number of obstacles. In all three scenarios, as one would expect, the plots denoting the exploration time of Brick&Mortar and MDFS cross over towards the middle of the x-axis, since Brick&Mortar is faster with few obstacles, but becomes inefficient with many obstacles. In the *Collapsed* scenario (Figures 4.2C and 4.2D), we observe an unexpected behavior: the exploration time of Brick&Mortar is far lower than its visiting time. A careful study of this case revealed that Brick&Mortar agents were able to explore the area quite fast (thus achieving the exploration objective), but they interfered with each other whilst trying to resolve a large number of overlapping loops around obstacles. Thus, they required a much longer period to mark all cells as *visited*.

The HybridExploration algorithm has lower exploration and visiting times than the others in all three scenarios, and for varying numbers of obstacles. The reason is that it combines the ability of Brick&Mortar to quickly mark cells as *visited* when there are no obstacles, with the ability of MDFS to resolve loops, when there are many obstacles. It handles the presence of loops very well thanks to the virtual agents which are able to close them very quickly after they are created, while the physical agents are free to explore the remaining part of the area as fast as possible.The comparative benefits of HybridExploration are more pronounced in the Series scenario (Figures 4.2E and 4.2F), where it is up to to 50% faster than Brick&Mortar and MDFS.

Effect of area size: The next question that we address is whether the three proposed algorithms scale gracefully as we increase the number of cells in the area. A comparison of the four algorithms in three different scenarios is depicted in Figure 4.3. As one would expect, as we increase the area size, the exploration and termination times increase for all three algorithms in all scenarios. Let's take a closer look at the behavior of MDFS and Brick&Mortar. In terms of exploration time, small areas favour MDFS over Brick&Mortar, whereas large areas favour MDFS. The two algorithms meet towards the middle of the x-axis in all three graphs. The reason is that MDFS typically traverses each cell more

than once, whereas Brick&Mortar only once unless it has to resolve loops. The visiting times of the two algorithms is very close to the respective exploration times, except in the case of the *Collapsed* scenario where Brick&Mortar takes much longer to achieve the termination objective than to achieve the exploration objective. The reason for this gap is explained above in our discussion of the effect of obstacles.

HybridExploration, which combines the strengths of MDFS and Brick&Mortar, outperforms both of them in all three scenarios, and scales gracefully (sub-linearly) with the area size. In the Series scenario (Figures 4.3E and 4.3F), where the loops around obstacles are longer than in the other two scenarios, the ability of HybridExploration to close loops using virtual instead of physical agents gives it a significant advantage over Brick&Mortar and MDFS.

Effect of rooms: Figure 4.4 shows the effect of varying the number of rooms in the area while maintaining the same number of cells, and thus the same area size. It is peculiar how Brick&Mortar performs poorly in terms of visiting time, when there are no rooms at all (open space area). This can be explained by observing that the loops are not bound within a room but the agents can build loops which are as large as the whole area, and therefore the time needed to close them is much longer than in cases where a loop cannot be larger than the size of a room. In the *Collapsed* scenario (Figures 4.4C and 4.4D), however, an increase in the number of rooms slows down Brick&Mortar. The reason is that in this particular scenario, rooms have many collapsed walls (doors), and more rooms effectively introduce more loops that are difficult and time-consuming to resolve.

The behavior of MDFS is also interestingly different depending on the scenario. In the *Office* and *Collapsed* scenarios (Figures 4.4A, 4.4B, 4.4C and 4.4D), the performance of MDFS both in terms of exploring and visiting times hardly depends on the number of rooms. We observe a small decrease in the exploration and visiting times of MDFS, simply because some of the cells that were previously *unexplored* are now *wall* cells forming the frames of rooms; these wall cells do not require to be traversed and marked as *explored* or *visited*. The behavior of MDFS in the *Series* scenario (Figures 4.4E and 4.4F) is rather unexpected. MDFS is slowed down by the presence of rooms although with more rooms there are less cells to cover. We think that this is because one of the strengths of the MDFS algorithm is that the agents can build trees of *explored* cells, with several branches spanning different rooms in the scenario, and process them at the same time. This parallelisation though is not possible in the *Series* scenario, where the rooms form a chain and need to be explored one after the other, without the possibility to explore more than one at the same time. The agents running MDFS therefore need firstly to mark the cells in the room as *explored*, then traverse them again to mark them as *visited* and finally move on to the next room, without efficiently using all the agents in a parallel way during the exploration. The Brick&Mortar and HybridExploration algorithms on the contrary can directly mark every cell as *visited*, so that, although they cannot explore more rooms at the same time in parallel, the overall exploration and visiting times are much less than

the MDFS ones.

The HybridExploration algorithm outperforms the other two algorithms in all scenarios; again, the *Series* scenario presents the most interesting case, where the benefits of HybridExploration are more pronounced.

Summary of results: The HybridExploration algorithm outperforms the three competing approaches (Ants, MDFS and Brick&Mortar) in terms of exploration and visiting time, in all three scenarios (*Office*, *Collapsed*, and *Series*), and for a wide range of area sizes, obstacle, and room numbers. The benefits of HybridExploration compared to Ants, MDFS and Brick&Mortar are far more obvious in the *Series* scenario. In general, HybridExploration is faster because it combines the strengths of all the competing approaches. First, like MDFS it adopts a depth-first-search approach to resolve loops around obstacles, but using virtual agents instead of physical agents. Second, like Brick&Mortar, agents running HybridExploration traverse each cell approximately once before they mark it as *visited*. Finally, a variation of the Ants algorithm is used to let the physical agents escape from loop. Interestingly, whereas Brick&Mortar tends to traverse cells multiple times when we introduce obstacles, HybridExploration is more robust when obstacles are present, because it resolves loops around them much faster with the aid of virtual agents.

4.3 Real Experiment

To study the feasibility of the approach in a real environment, we tested the algorithm by using a mobile robot which deployed Tmote Sky [48] sensors on the floor of a large open-plan office area. We used a Lego Mindstorm NXT [49] robot and several Tmote Sky sensors, running the Contiki Operating System [50]. The aim of the experiment was to prove that a robot can deploy sensors on the ground, use them to store information about visited and explored cells, and to correct the odometry errors caused by the inaccuracy of its movement.

We used a very simple off-the-shelf robot to prove that no special purpose hardware is needed when we integrate the navigation system (based on the robot movements) with the deployed sensor network. Figure 4.5 shows a view of the robot close to a mote that is assumed to be dropped on the floor by the robot itself. The robot does not yet have a device able to autonomously place the motes on the floor, so the motes were placed by hand after the robot stopped in a certain position. Moreover, since the robot does not have 802.14 wireless capabilities, we attached another mote on top of it and used a laptop (with another mote acting as a gateway) as a bridge between 802.14 and bluetooth messages, in order to have a direct channel between the robot and the sensor.

Performing this and other experiments, we found out that the received signal strength heavily depends on factors like the relative orientation of the motes, their height from the

floor, the material of the floor, and the obstacles in the environment (the line of sight). In fact, in the literature ([12], [13], [15], [16], [17], and [18]) it is widely accepted that radio propagation is (i) non-isotropic (i.e., the received signal, at a given distance from the sender, is not the same in all directions), it has (ii) non-monotonic distance decay (i.e., lower distance does not always mean better link quality), and (iii) the communication is based on asymmetrical links (i.e., if A hears B, it cannot be assumed that B hears A). For all these reasons, in our experiment we limited the transmitting power to the minimum and deployed the motes upside down on the floor (as shown in Figure 4.5) so that their transmission range was limited, and most of the radio waves were received directly and not after several bounces on the obstacles in the environment. Furthermore, if we assume that the positions of the deployed motes do not change (once dropped, the motes keep their mutual position and orientation), then we can also assume the following: if the robot is at position A at times t_1 and t_2 (with $t_2 = t_1 + n$), then it will receive a signal from the same motes with a very similar signal strength in both cases, unless the environment heavily changes between t_1 and t_2.

In an ideal situation, the robot could store a matrix with all the *explored*, *visited*, and *wall* cells that it marked, and it could be able to navigate through them simply by measuring the travelled distance in each direction and the number of turns made (odometry). Unfortunately, in a real situation, each time the robot turns or travels, a small error is introduced, and after a while the real position of the robot could be completely different from the one which it calculated. In Figure 4.6, we show an example describing the odometry errors introduced by the robot during the exploration of the floor of a large open-plan office area. We also show how the robot is able to partially correct them during its exploration. In the same figure, *n1, n2, ..., n8* are the identifiers of the motes deployed on the floor by the robot, while the labels on the edges (*1, 2, ..., 9*) represent the steps followed by the robot during its movement. For instance, in Figure 4.6 the robot departs from the starting point position and every time it performs a movement, it stops, it drops a mote on the floor, and it broadcasts a message using the maximum transmission power, so that its message is received by all the sensors which are in range. Each time a sensor receives a message, it sends other two messages back to the robot, each one with a different transmission power (e.g., one message with the minimum and the other one with a slightly higher transmission power). In this way, if the robot receives back two messages from a sensor, it recognises to be on the same cell (area) in which the sensor is. On the other hand, if it just receives messages with the highest transmission power, then it understands that no sensor is really close to it (or better the robot itself is not in the same cell in which a sensor has been previously deployed). Moreover, the time interval between the two broadcasted messages has been randomly varied to avoid collisions. In this way, every time the robot performs a step, it can build a list of received messages containing the following information:

- the identifier of the sender (i.e., *n1, n2, ..., n8*);

- the received signal strength indicator (RSSI);
- the transmission power used by the sender to transmit the message.

Consequently, the robot can associate a list of received messages to each position in which it previously deployed a mote. For example, as soon as the robot drops the mote $n2$ on the floor, then it broadcasts a message expecting to receive both two messages back from the same mote $n2$ (with the smallest and the highest transmission power), and one message back only from the first deployed mote $n1$ (with the highest transmission power). This situation allows the robot to recognise if there is an already deployed mote in the cell in which it is.

To correct an odometry error, every time the robot visits a cell in which it already deployed a sensor, the broadcast is repeated and the received messages should be similar to the ones received when the robot was standing in the same cell before. If this is not the case, then the robot tries to infer the actual position relatively to the calculated erroneous one. For example, while performing the step *3*, an odometry error occurs. This means that the robot, instead of following the same step *2* while going back, performed a slightly different path (step *3*) without knowing it. To recognise this event, once stopped the robot broadcasts a message and it receives from sensor $n2$ one message only with the highest transmission power. In this way, the robot realises that it is not standing in the same cell in which it previously deployed sensor $n2$ (as expected). In order to correct its position and to move toward mote $n2$, it uses the received signal strength of the message sent back by sensor $n1$ and if this signal strength is decreased respect to the one it was stored before while it was close to sensor $n2$, then the robot moves directly toward the right direction (so toward $n2$). For this reason, to correct the odometry error the robot performs step *3'* to get closer to sensor $n2$. The entire message exchange procedure is then repeated until the robot receives not only one but two messages back from sensor $n2$ (demonstrating that the robot stands in the cell in which the sensor is). The same process is repeated to solve the odometry error occurring during step *7*. The correction is done by performing the additional step *7'*. To generalise, if more messages (or messages with higher signal strength) are received from motes within a certain direction, and less messages (or messages with lower signal strength) are received from motes within another direction, the robot travels a smaller distance to the latter direction, and then rebroadcasts the message. This process is repeated until the list of the latest received messages is similar enough to the list which was received when the robot firstly deployed the mote in the cell, with the addition of the messages received from the new motes which have been deployed in the environment afterwards. Figure 4.7 shows both how the environment is seen by the robot during the exploration process and how the motes are deployed accordingly to the actual robot position.

Figure 4.8 shows the environment that the robot (R representing the current robot position) will discover during the exploration. We assume that the robot comes from a different

room and it simply continues its exploration toward unexplored areas, thus a corridor of cells, already marked as *explored* (as seen in Figure 4.8), is left behind. In this way, we just provide an example of how the robot opens a corridor of *explored* cells successively using them to go back after having finished to mark as *visited* the overall unexplored environment. Moreover, Figure 4.9 describes how the cells, in which the environment is assumed to be subdivided, are marked as *visited* or *explored* according to the Physical Agent Protocol rules.

In conclusion, with this experiment we mainly proved that a robot can potentially deploy sensors on the ground, use them to store information about visited and explored cells, and correct the odometry errors caused by the inaccuracy of its movement. However, we also found that the signal strength alone cannot be considered accurate enough so as to use it to localize the robot. Nevertheless, with this experiment we proved that, integrating the signal strength with the odometry of the robot, a better estimation of the relative position of the robot itself respect the other stationary nodes can be found, and eventually the robot can identify the cell in which it is located.

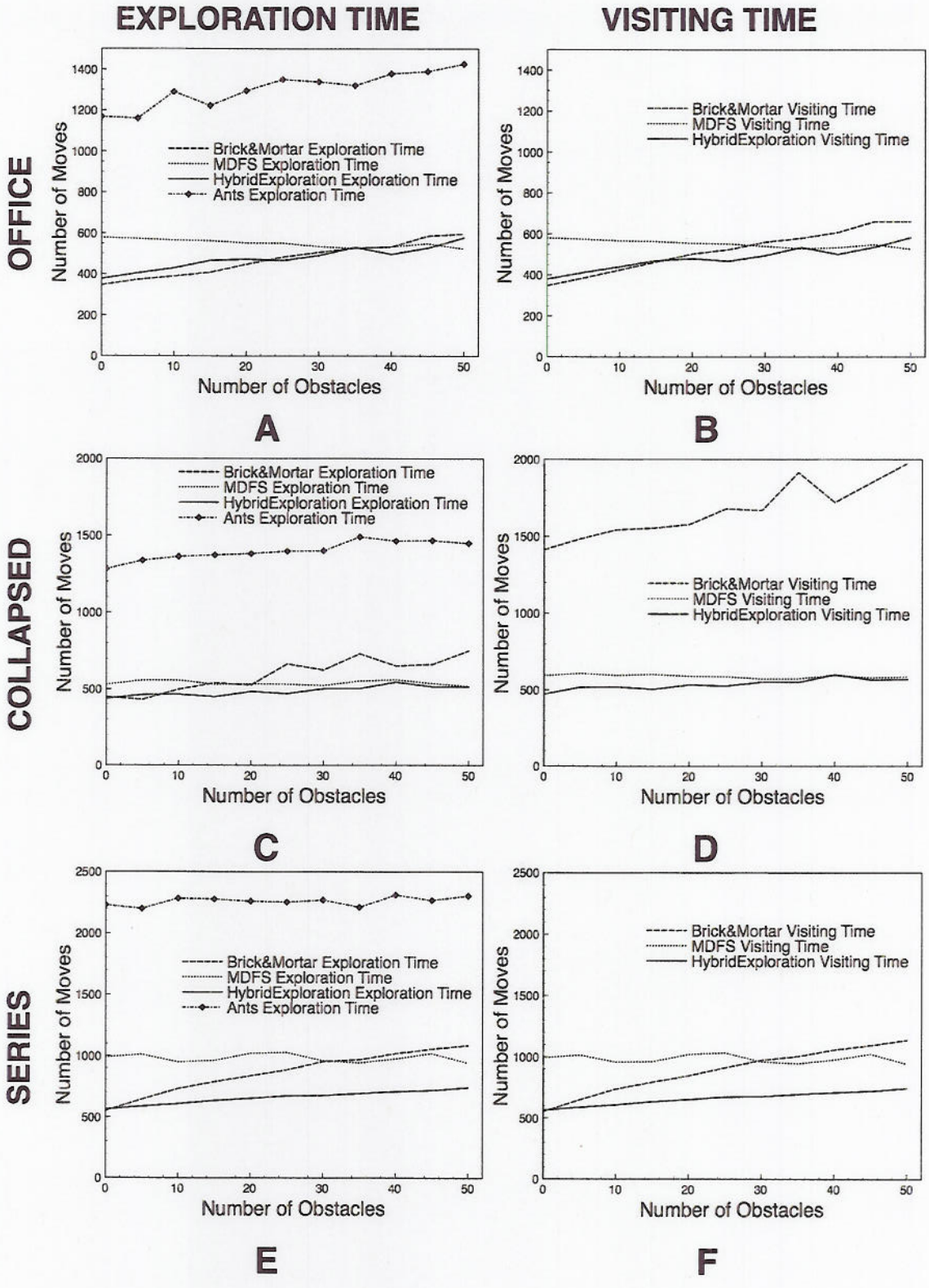

Figure 4.2: Effects of changing the number of obstacles.

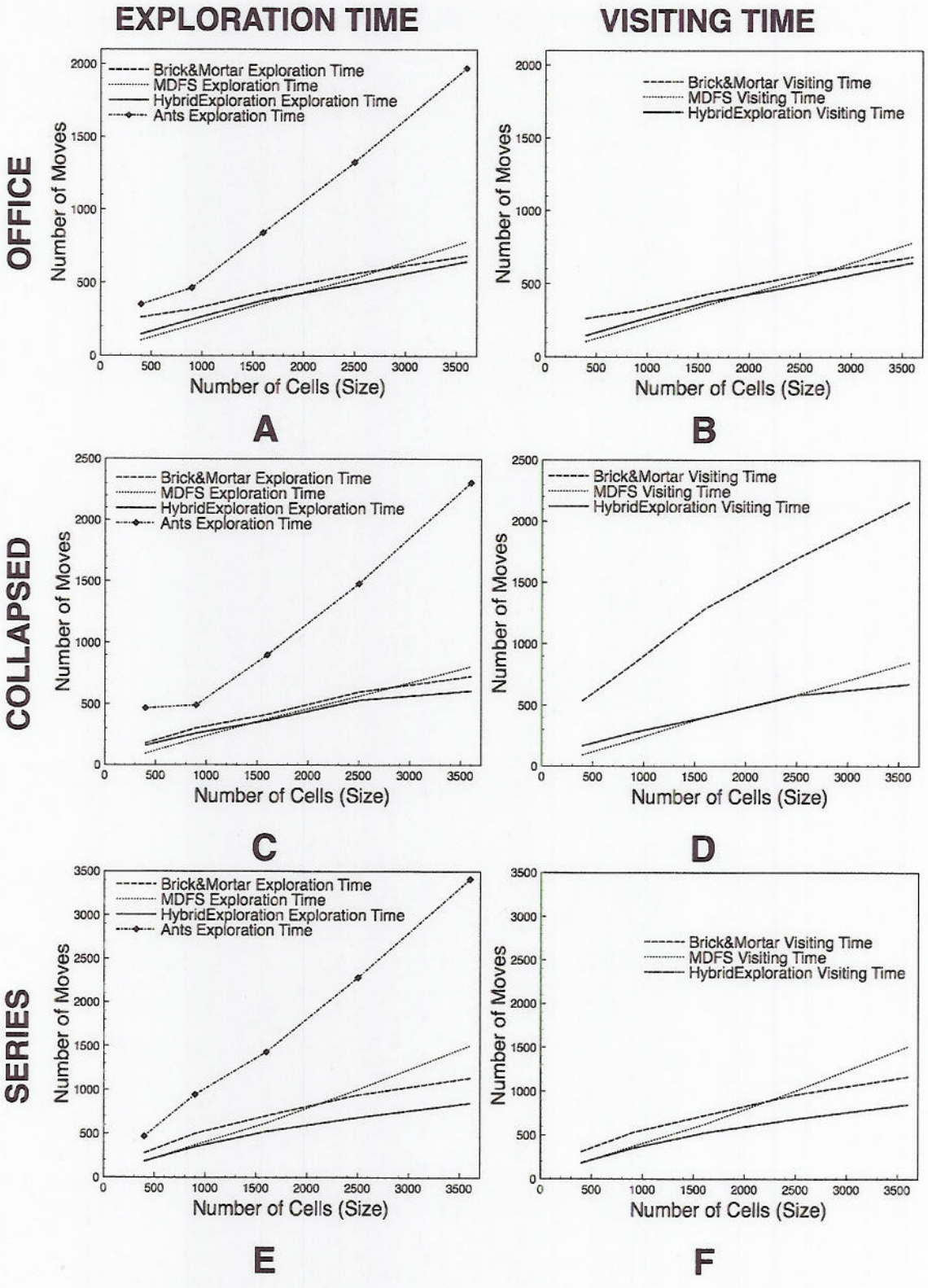

Figure 4.3: Effects of changing the size of the map.

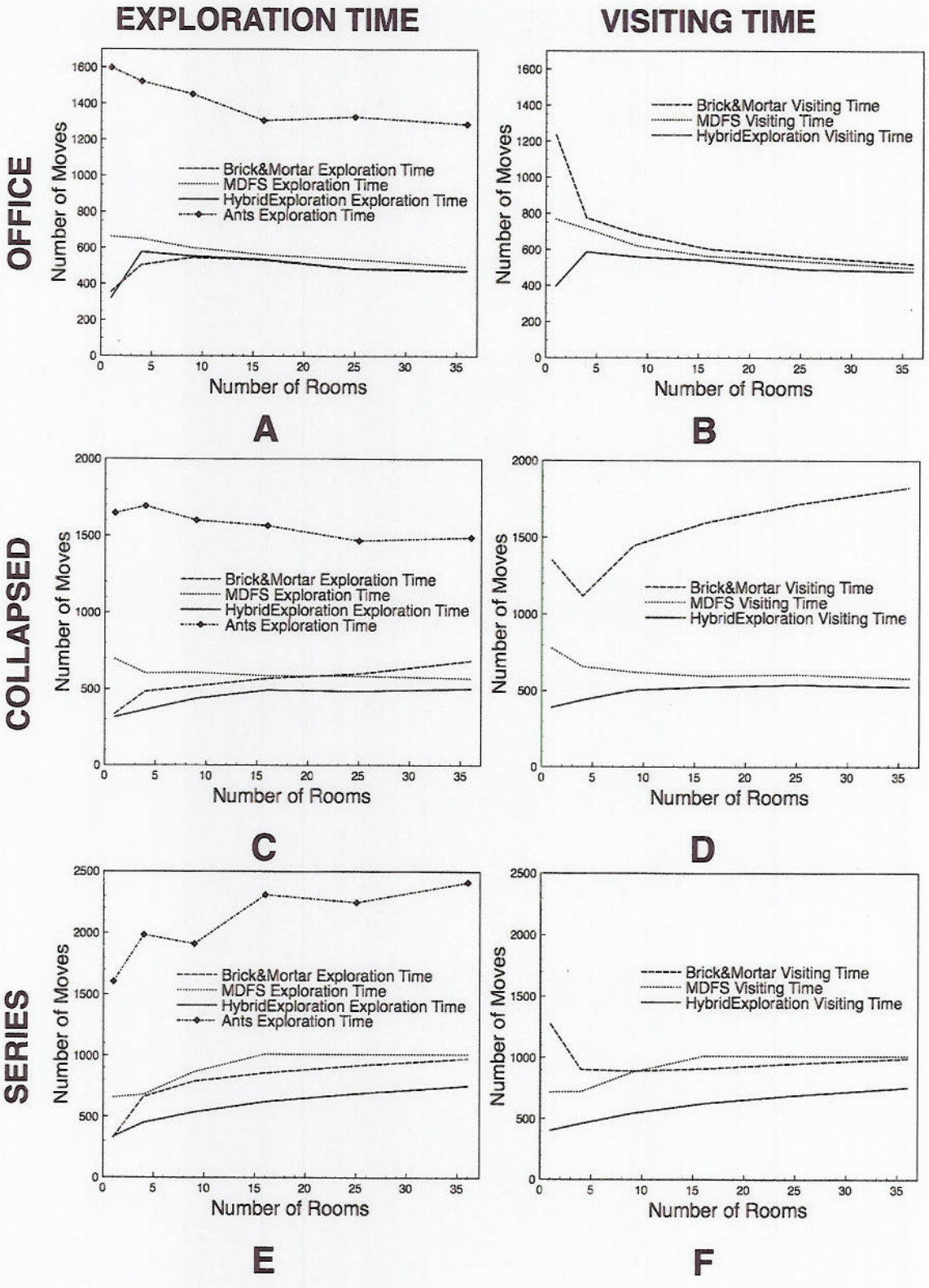

Figure 4.4: Effects of changing the number of rooms.

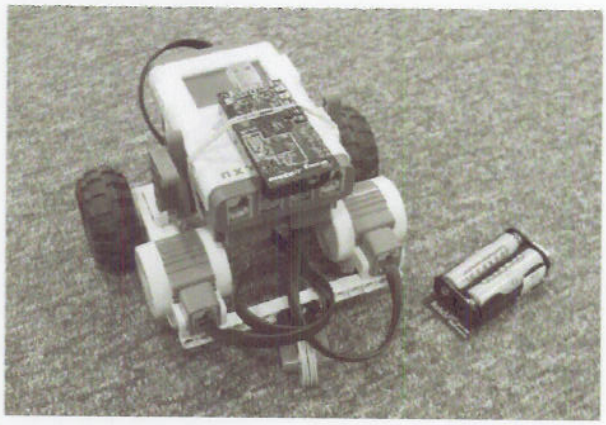

Figure 4.5: View of the robot close to a mote. The mote is upside down to limit its transmission range and suit the needs of the algorithm.

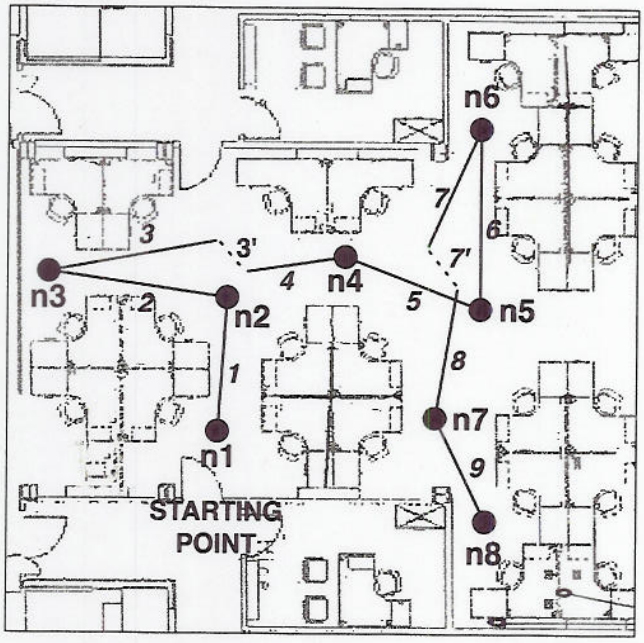

Figure 4.6: Map of the floor with the trail followed by the robot. Both the deployed motes (*n1, n2, ..., n8*) and the steps (*1, 2, ..., 9*) followed by the robot during its movement can be seen in the figure.

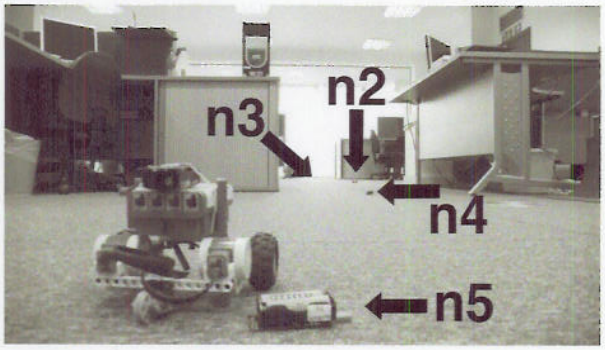

Figure 4.7: The robot during the exploration. Part of the deployed motes (*n5, n4, n3, n2*) can be seen in the figure.

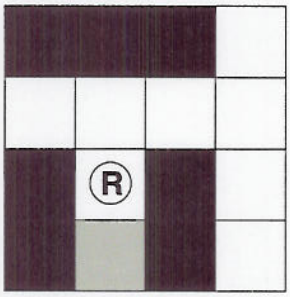

Figure 4.8: Map of the environment that will be discovered by the robot R. A corridor of *explored* (grey) cells is left behind to allow the robot to go back and continue the exploration.

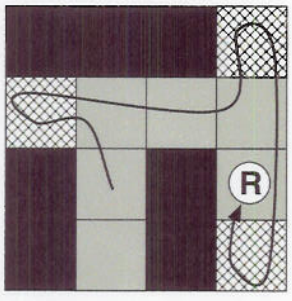

Figure 4.9: Map as seen by the robot R after a partial exploration has been completed. Both *explored* (grey) and *visited* (patterned) cells are shown.

Chapter 5

Summary

The first step of our work was to provide an extensive literature review of existing approaches to autonomous exploration of unknown terrains, and a discussion of their assumptions, limitations, and weaknesses (Chapter 2). We then clearly identified the objective of our research, i.e. the creation of novel algorithms for the multi-agent autonomous exploration of indoor unknown terrains in the minimum amount of time (Chapter 1). We specified our assumptions, i.e. the area map is unknown, there is no existing network infrastructure, long-range wireless communication is unreliable, and nodes are not location-aware (no GPS support is provided). We then introduced three novel exploration algorithms, namely Multiple Depth First Search, Brick&Mortar, and HybridExploration algorithms (Chapter 3) and compared them with the existing Ants algorithm [5] (Chapter 2). Finally, we carefully analysed the strengths, and weaknesses of the proposed and existing algorithms, and set up a simulation environment to evaluate their performance under various network conditions (Chapter 4).

Bibliography

[1] E. Ferranti, N. Trigoni and M. Levene. *Brick&Mortar: An On-Line Multi-Agent Exploration Algorithm.* In *ICRA07: Proceedings of the 2007 IEEE International Conference on Robotics and Automation* pages 761–767. IEEE press April 2007.

[2] I. Rekleitis, G. Dudeck and E. Milios. *Multi-robot Exploration of an Unknown Environment, Efficiently Reducing the Odometry Error.* In *IJCAI97: Proceedings of the 1997 International Joint Conference on Artificial Intelligence* pages 1340–1345. Morgan Kaufmann August 1997.

[3] C. Icking, T. Kamphans, R. Klein and E. Langetepe. *Exploring Simple Grid Polygons.* In *COCOON05: Proceedings of the 2005 Annual International Conference of Computing and Combinatorics* pages 524–533. Springer Berlin / Heidelberg August 2005.

[4] N. Hazon, F. Mieli and G. A. Kaminka. *Towards Robust On-line Multi-Robot Coverage.* In *ICRA06: Proceedings of the 2006 IEEE International Conference on Robotics and Automation* pages 1710–1715. IEEE press May 2006.

[5] J. Svennebring and S. Koenig. *Building Terrain-Covering Ant Robots: A Feasibility Study.* Autonomous Robot **16** (3), 313–332 (May 2004).

[6] H. Choset. *Coverage for robotics - A survey of recent results.* Annals of Mathematics and Artificial Intelligence **31**, 113–126 (2001).

[7] S. Koenig and Y. Liu. *Terrain Coverage with Ant Robots: a Simulation Study.* In *AGENTS01: Proceedings of the 2001 ACM International Conference on Autonomous Agents* pages 600–607. ACM Press May 2001.

[8] A. Kleiner, J. Prediger and B. Nebel. *RFID Technology-based Exploration and SLAM for Search And Rescue.* In *IROS06: Proceedings of the 20062006 IEEE/RSJ International Conference on Intelligent Robots and Systems* pages 4054–4059. IEEE Press October 2006.

[9] X. Zheng, S. Jain, S. Koenig and D. Kempe. *Multi-Robot Forest Coverage.* In *IROS05: Proceedings of the 2005 IEEE International Conference of Intelligent Robots and Systems* pages 3852–3857. IEEE press August 2005.

[10] N. Agmon, N. Hazon and G. A. Kaminka. *Constructing Spanning Trees for Efficient Multi-Robot Coverage, booktitle = ICRA06: Proceedings of the 2006 IEEE International Conference on Robotics and Automation.* pages 1698–1703. IEEE press May 2006.

[11] N. Hazon and G. A. Kaminka. *Redundancy, Efficiency, and Robustness in Multi-Robot Coverage.* In *ICRA05: Proceedings of 2005 IEEE International Conference on Robotics and Automation* pages 735–741. IEEE press April 2005.

[12] D. Kotz, C. Newport, R. S. Gray, J. Liu, Y. Yuan and C. Elliott. *Experimental Evaluation of Wireless Simulation Assumptions.* Technical Report Technical Report TR2004-507, Dartmouth College Computer Science June 2004.

[13] D. Aguayo, J. Bricket, S. Biswas, G. Judd and R. Morris. *Link-level Measurements from an 802.11b Mesh Network.* In *SIGCOMM04: Proceedings of the 2004 ACM Conference of the Special Interest Group on Data Communication* pages 121–132. ACM press August 2004.

[14] J. Park, S. Park, D. Kim, P. Cho and K. Cho. *Experiments on radio interference between wireless LAN and other radio devices on a 2.4 GHz ISM band.* In *VTC03: Proceedings of the 2003 IEEE Semiannual Vehicular Technology Conference* pages 1798–1801. IEEE Press April 2003.

[15] D. Kotz, C. Newport and Elliot C. *The mistaken axioms of wireless-network research.* Technical Report Technical Report TR2003-467, Dartmouth College Computer Science July 2003.

[16] J. Zhao and R. Govindan. *Understanding Packet Delivery Performance In Dense Wireless Sensor Networks.* In *SenSys03: Proceedings of the 2003 ACM Conference on Embedded Networked Sensor Systems* pages 1–13. ACM press November 2003.

[17] A. Cerpa, N. Busek and D. Estrin. *SCALE: A tool for Simple Connectivity Assessment in Lossy Environments.* Technical Report Technical Report CENS-21, UCLA September 2003.

[18] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin and S. Wicker. *Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks.* Technical Report Technical Report CSD-TR 02-0013, UCLA February 2002.

[19] D. Hähnel, W. Burgard, D. Fox, K. Fishkin and M. Philipose. *Mapping and Localization with RFID Technology.* In *ICRA04: Proceedings of 2004 IEEE International Conference on Robotics and Automation* pages 1015–1020. IEEE Press April 2004.

[20] W. Burgard, M. Moors, C. Stachniss and F. Schneider. *Coordinated multi-robot exploration.* IEEE Transactions on Robotics **21** (3), 376–378 (June 2005).

[21] M. A. Batalin and G. S. Sukhatme. *The Analysis of an Efficient Algorithm for Robot Coverage and Exploration based on Sensor Network Deployment.* In *ICRA05: Proceedings of 2005 IEEE International Conference on Robotics and Automation* pages 3478–3485. IEEE press April 2005.

[22] C. S. Kong, N. A. Peng and I. Rekleitis. *Distributed Coverage with Multi-Robot System.* In *ICRA06: Proceedings of the 2006 IEEE International Conference on Robotics and Automation* pages 2423–2429. IEEE press May 2006.

[23] B. Yamauchi. *Frontier-Based Exploration using Multiple Robots.* In *AGENTS98: Proceedings of the 1998 ACM International Conference on Autonomous Agents* pages 47–53. ACM press May 1998.

[24] A. Howard, L. E. Parker and G. S. Sukhatme. *Experiments with a Large Heterogeneous Mobile Robot Team: Exploration, Mapping, Deployment and Detection.* The International Journal of Robotics Research **25** (5–6), 431–447 (December 2006).

[25] W. Burgard, M. Moors, D. Fox, R. Simmons and S. Thrun. *Collaborative Multi-Robot Exploration.* In *ICRA00: Proceedings of the 2000 IEEE International Conference on Robotics and Automation* pages 476–481. IEEE press April 2000.

[26] M. A. Batalin and S. G. Sukhatme. *Spreading Out: A Local Approach to Multi-robot Coverage.* In *DARS02: Proceedings of the 6th International Symposium on Distributed Autonomous Robotics Systems* pages 373–382 June 2002.

[27] RoboCup Rescue. http://www.isd.mel.nist.gov/projects/USAR/competitions.htm.

[28] M. Kadous, R. Sheh and C. Sammut. *Controlling Heterogeneous Semi-autonomous Rescue Robot Teams.* In *SCM06: Proceedings of the 2006 IEEE International Conference on Systems, Man, and Cybernetics* pages 3204–3209. IEEE Press October 2006.

[29] *Walking Machine ETS.* http://wm.etsmtl.ca/website2006/en/index.php.

[30] *Bremen Rescue Walkers.* http://aimee.informatik.uni-bremen.de/index.php.

[31] M. Albrecht, T. Backhaus, S. Planthaber, H. Stoeppeler, D. Spenneberg and Kirchner F. *AIMEE: A Four Legged Robot for RoboCup Rescue.* In *CLAWAR05: Proceedings of the 8th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines* pages 1003–1010. Springer September 2005.

[32] *Deutschland1.* http://www.inf.uos.de/kbs/deutschland1.html.

[33] K. Lingemann, A. Nüchter, J. Hertzberg, O. Wulf, B. Wagner, K. Pervölz, K. Surmann and T. Christaller. *RoboCupRescue2006 – Robot League, Deutschland1(Germany).* In *Rescue Robot League Competition, (CDROM Proceedings)* June 2006.

[34] A. Birk, S. Markov, I. Delchev and K. Pathak. *Autonomous Rescue Operations on the IUB Rugbot.* In *SSRR06: Proceedings of the 2006 IEEE International Workshop on Safety, Security, and Rescue Robotics.* IEEE Press August 2006.

[35] A. Kleiner, C. Dornhege, R. Kuemmerle, M. Ruhnke, B. Steder, B. Nebel, P. Doherty, M. Wzorek, P. Rudol, G. Conte, S. Durante and D. Lundstrom. *RoboCupRescue - Robot League Team RescueRobots Freiburg (Germany).* In *TDP and Poster Real Robot, Rescue Robot League, (CDROM Proceedings).* IEEE Press June 2006.

[36] Resko Team. http://robots.uni-koblenz.de/.

[37] Aryaak. http://www.azad.ac.ir/.

[38] MRL (Naji). http://www.iust.ac.ir/IUSTEntry/Default.asp.

[39] Resquake. http://saba.kntu.ac.ir/eecd/taghirad/resquake.html.

[40] Alcor. http://www.dis.uniroma1.it/ pirri/alcor.htm.

[41] C-Rescue. http://www.chukyo-u.ac.jp/eng/index.html.

[42] NIIT Blue. http://www.niit.ac.jp/english/english-top.htm.

[43] Nutech-R. http://www.nagaokaut.ac.jp/e/index.html.

[44] Toin Pelican. http://www.cc.toin.ac.jp/UNIV/english/index.html.

[45] Roscue. http://www.robhaz.com/eng/default.asp.

[46] RFC Uppsala. http://www.robocup.it.uu.se/magne.

[47] Good Samaritan. http://www.engr.colostate.edu/me/RAMlab/goodsam/index.html.

[48] Tmote Sky Platform. http://www.moteiv.com/.

[49] Lego Mindstorm NXT. http://mindstorms.lego.com/.

[50] The Contiki Operating System. http://www.sics.se/contiki/.

[51] H. Zhou and S. Sakane. *Mobile robot localization using active sensing based on Bayesian network inference.* Robotics and Autonomous Systems **55** (4), 292–305 (April 2007).

[52] N. Priyantha, H. Balakrishnan, E. Demaine and S. Teller. *Mobile-Assisted Localization in Wireless Sensor Networks.* In *INFOCOM 2005: 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* pages 172 – 183. IEEE press March 2005.

[53] J. Bae and S. Lee. *Active Sensing based Mobile Robot Exploration.* In *Proceedings of the 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics* pages 934 – 939. IEEE press July 2005.

[54] M. Batalin, G. Sukhatme and M Hattig. *Mobile Robot Navigation using a Sensor Network.* In *ICRA04: Proceedings of the 2004 IEEE International Conference on Robotics and Automation* pages 636–642. IEEE press April 2004.

[55] H. Zhou and S. Sakane. *Sensor Planning for Mobile Robot Localization - A hierarchical approach using Bayesian network and particle filter.* In *Proceedings of the 2004 IEEE International Conference on Robotics and Biomimetics* pages 540 – 545. IEEE press August 2004.